Technische Universität Darmstadt
Fachgbereich Elektrotechnik und Informationstechnik
Fachgebiet Mikroelektronische Systeme

Optimal Design of Fixed-Point and Floating-Point Arithmetic Units for Scientific Applications

Surapong Pongyupinpanich

PhD Thesis, March 2012

Institute of Microelectronic Systems
Technische Universität Darmstadt
Merckstr. 25, D-64283 Darmstadt, Germany

Optimal Design of Fixed-Point and Floating-Point Arithmetic Units for Scientific Applications

Vom Fachbereich 18
Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

M.Eng.
Surapong Pongyupinpanich
geboren am 17. Juni 1976
in Prachinburi, Thailand

Referent:	Prof. Dr. Dr. h. c. mult. Manfred Glesner *Technische Universität Darmstadt*
Korreferent:	Prof. Dr.-Ing. Michael Hübner *Ruhr-Universität Bochum*
Tag der Einreichung:	20.04.2012
Tag der mündlichen Prüfung:	17.08.2012

D17

Darmstädter Dissertationen 2012

Acknowledgment

This thesis is based on my work that I have started in March 2008 at *Fachgebiet Mikroelektronische Systeme*, *Institute für Datentechnik, Fachbereich Elektrotechnik und Informationstechnik*, *Technische Universittät Darmstadt* as a research assistant with a scholarship from *Ramkamhang University*, Thailand. Therefore, I would like to thank *Ramkamhang University* for awarding me the scholarship to pursue my doctoral degree. Special thanks are due to my advisor Prof. Dr. Dr. h.c. mult. Manfred Glesner for his advices, guidance and nice working environment. The colourful working atmosphere at his research institute reflects his internationally open personality, quality and care, from which I have benefited.

As my co-advisor, I express my acknowledgment to Prof. Dr.-Ing. Michael Hübner for his supports and advices. I would like to thank Prof. Dr.-Ing. habil. Dr. h.c. Andreas Binder, Prof. Dr.-Ing. Harald Klingbeil and Prof. Dr.-Ing. Hans Eveking for spending time to be the committee of my oral examination. I would like to thank acknowledge Assoc. Prof. Dr. Somsak Choomchuay (Ph.D. DIC) for his recommendation to pursue higher educational degree that is useful for my academic career.

I gratefully acknowledge Dr.-Ing. Fizal Arya Samman who has spent much time with me to discuss and share technical knowledge about improvement of floating-point arithmetic topic. I thank Prof. Dr.-Ing. Peter Zipf for the time to discuss about my topic in the particular area of system-level design, as well as to all anonymous reviewers of my journal and conference papers for the positive critics and suggestions. Many thanks are due to the former staff members at *Fachgebiet Mikroelektronische Systeme*, Dr.-Ing Andre Guntoro, Dr.-Ing Massoud Momeni, Dr.-Ing Oana M. Cobianu, Dr.-Ing Heiko Hinkelmann, Dr.-Ing. Petru Bacinschi, Dr.-Ing Ping Zhao, Dr.-Ing Leandro Möller as well as the current staff members, Ramkumar Ganesan, Sebastian Pankalla, Francois Philipp, Christopher Spies, Elvio Dutra e Silva and Enkhbold Ochirsuren for the friendship and cooperation.

I would also like to express my appreciation to the staff members at *Fachgebiet Integrierte Elektronische Systeme*, the head of the institute Prof. Dr.-Ing Klaus Hofmann and his research and teaching assistant staffs, Harish Balasubramaniam, Yuan Fang, Ashok Jaiswal, Mareiki Kaloumenos, Jing Ning, Muhammad Saif, Alex Schönberger, Lufei Shen, Boris Traskov and Haoyuan Ying. My acknowledgments are granted to Andres Schmidt and Roland Brand for helping me in many things about software and hardware matters, and to Silvia Hermann and Iselona Klenk for helping me in many administrative matters.

I express my obligation to all my supervised students who have made nice cooper-

ations with me in the framework of bachelor/master/diploma thesis. Thank you very much also to Gordon Smith for proof-reading and correcting the written-english thesis. My stay in Darmstadt is enhanced by many friends and all member staffs at Banthai Restaurant. For the fruitful friendships, I appreciate all my colleagues from Thailand, Germany and from all other countries that I could not mention them in this page.

I deeply acknowledged all my teachers in my primary school, secondary school and my high school in Prachinburi, Nonthaburi, as well as my lecturers at *King Mongkut's Institute of Technology Ladkrabang* in Bangkok for providing me with basic and advanced knowledge. Many thanks are also given to all teaching, technical and administrative staff members at *Ramkamhang University* in Bangkok for their helps and administrative supports.

From the depth of my heart, I am grateful to my lovely mother and my lovely father for their patience to advice and educate me. Their love, care, dedication and their long life educational supports connot be expressed with words. I pray for them to be blessed, and their prayer is a strong motivation for me to make them proud. I would like to thank my younger sister Narumol Pongyupinpanich and her husband as well as my friend, Mongkol Jalerntam, Sakkarn Kaewket, Thawatchai Wachiradusit and Rachot Keatbunjon for their supports and for taking care well my lovely parent during I am staying in Germany.

Darmstadt, August 2012
Surapong Pongyupinpanich

Abstract

The challenge in designing a floating-point arithmetic co-processor/processor for scientific and engineering applications is to improve the performance, efficiency, and computational accuracy of the arithmetic unit. The arithmetic unit should efficiently support several mathematical functions corresponding to scientific and engineering computation demands. Moreover, the computations should be performed as fast as possible with a high degree of accuracy. Thus, this thesis proposes algorithm, design, architecture, and analysis of floating-point arithmetic units particularly for scientific and engineering applications which can be implemented in *VLSI*.

Generally, performance improvements and time efficiency with hardware can be considered from the output rate and the computational latency which is the number of generated outputs per second (output/sec) and the computational times. The output rate can be increased by clock rate whereas the design and architecture of the hardware can improve the computational time, which is mostly focused on engineering practice. Obviously, in order to achieve the highest performance, the design will be based on pipeline architecture. Nevertheless, for any hardware arithmetic unit, not only the performance and time efficiency have to be examined, but also the computational accuracy and stability of the computational results have to be taken into account. Therefore, the floating-point arithmetic units introduced in this dissertation will be considered in their design and architecture based on pipeline, and an analysis of the hardware trade-off between the *VLSI* areas of complexity and computational latency. Meanwhile, the floating-point data representation is employed to improve and stabilize the computational result and accuracy of the arithmetic unit at runtime.

The arithmetic units from a hardware point of view can be classified into two groups depending on hardware-based algorithms, i.e. the basic arithmetic unit and the advanced arithmetic unit. The basic arithmetic unit consists of two types of operations corresponding to the number of input operands, i.e. standard operations and non-standard operations. The standard operations are addition/subtraction and multiplication operations and the non-standard operations are product-of-sum and sum-of-product operations. The advanced arithmetic unit is frequently employed in scientific and engineering applications as elementary functions such as sine, cosine, hyperbolic sine, hyperbolic cosine, etc. The two classes of arithmetic units can be derived in hardware-based algorithmic form which is relatively easy for *VLSI* implementation and for analysis.

The binary-tree and partial linear methods are introduced to the leading-one-detection (*LOD*) and the integer multiplier in order to improve the performance of the floating-point standard and non-standard operators. The investigation and synthesis results that are based on the pipeline architecture show that both the proposed floating-point standard and nonstandard hardware-based algorithms can be simplified for VLSI implementation. Meanwhile, with the proposed *LOD* and the proposed integer multiplier, the floating-point standard and non-standard operators provide both high performance and time efficiency.

The advanced arithmetic functions are performed by the CORDIC algorithm, where the challenges of the CORDIC algorithm are to reduce computational latency and to improve computational accuracy. Therefore, two CORDIC methods, namely the double-rotation and triple-rotation, are proposed. Their performance, efficiency, and computational accuracy are measured, analysed, and compared with conventional CORDIC results using the Matlab/Simulink tools. The proposed CORDIC methods provide better performance, time efficiency, and computational accuracy than the conventional method, while at the same provided error constraints with few iterations. Similarly, with the same number of iterations, the proposed CORDIC methods present better computational accuracy than the conventional method. The unified micro-rotations of the proposed CORDIC methods are established and analysed in order to study the performance and efficiency based on several pipeline stages.

A high precision CORDIC algorithm, based on a unified micro-rotation of the proposed CORDIC methods, is introduced where the double-rotation and triple-rotation are applied for the normal-accuracy and high-accuracy mode, respectively. The high precision CORDIC core based on fixed-point representation is designed, implemented, and analysed. The synchronization between the floating-point standard unit, non-standard unit and the fixed-point elementary functional unit is demonstrated by floating-point arithmetic accelerator architecture and also by floating-point streaming processor architecture. The *Floating-to-Fixed* and *Fixed-to-Floating* algorithms are introduced for data conversion from floating-point to fixed-point representation and from fixed-point to floating-point representation.

Finally, the beam phase and magnitude detector that is employed in the closed-loop control system for heavy ion synchrotron application is used for verification of the proposed CORDIC methods. In the heavy ion synchrotron application, acceleration processes lead to beam signals with decreasing time periods for the pulses. Different modes of oscillation are possible. However, the current system deals with the simplest mode of oscillation, which is almost permanently presented, if no countermeasures are taken. The beam phase control system introduced here is dedicated to cases where all bunches are oscillating in phase. Therefore, the beam phase and magnitude detector is required to observe the beam oscillation for the closed-loop control system. The design of the digital phase and magnitude detector is modelled and simulated by *VHDL* on ModelSim. The simulation results based on the two patterns, *"Gap voltage"* and *"Beam position"* generated and captured from the mathematic model and the actual ion synchrotron

system, SIS18 at GSI Helmholtzzentrum Schwerionenforschung, are compared with the Matlab/Simulinks ideal results in order to verify the proposed CORDICs computation.

Kurzfassung

Beim Entwurf arithmetischer Prozessoren oder Koprozessoren für wissenschaftliches Rechnen liegt die Herausforderung darin, Rechenleistung, Effizienz, und numerische Genauigkeit zu maximieren. Derartige arithmetische Einheiten sollen mathematische Funktionen, die in wissenschaftlichen und technischen Anwendungen häufig benötigt werden, effizient unterstützen. Natürlich sollen sie Berechnungen so schnell wie möglich und mit hoher Genauigkeit durchführen. Die vorliegende Arbeit stellt deshalb Algorithmen und Architekturen für arithmetische Gleitkomma-Einheiten vor, die besonders für wissenschaftliches Rechnen geeignet sind. Die vorgestellten Architekturen sind zur Umsetzung in hochintegrierte (*VLSI*-)Schaltungen geeignet.

Allgemeine Kenngrößen zur Beurteilung arithmetischer Einheiten sind Durchsatz und Latenz. Der Durchsatz ist die pro Zeiteinheit verarbeitete Datenmenge, die Latenz die zur Verarbeitung eines Datums benötigte Zeit. Der Durchsatz kann gesteigert werden, indem die Taktrate einer synchron getakteten Schaltung erhöht wird, während die Architektur Einfluss auf die Latenz hat. Zum Erreichen höchster Rechenleistung werden *Pipeline*-Architekturen verwendet. Bei den in der vorliegenden Arbeit vorgestellten Architekturen handelt es sich deshalb stets um *Pipeline*-Architekturen und es wird ein Kompromiss (*Trade-Off*) zwischen der Komplexität einer Architektur und ihrer Latenz gesucht. Jedoch müssen bei der Beurteilung arithmetischer Einheiten nicht nur die Rechenleistung, sondern auch die numerische Genauigkeit und die Stabilität der implementierten Algorithmen betrachtet werden. In den betrachteten Architekturen kommt deshalb die Gleitkomma-Zahlendarstellung zum Einsatz um die numerische Genauigkeit und Stabilität zu verbessern.

Die in der vorliegenden Arbeit betrachteten arithmetische Einheiten werden abhängig von den implementierten Algorithmen in zwei Klassen eingeteilt, nämlich in einfache und fortgeschrittene Einheiten. Einfache Einheiten implementieren zwei Klassen von Operationen, nämlich Standard- und Nichtstandard-Operationen. Standard-Operationen sind Addition, Subtraktion und Multiplikation; Nichtstandard-Operationen sind *Product of Sums* und *Sum of Products*. Fortgeschrittene Einheiten implementieren zusätzlich Funktionen, die in wissenschaftlichen und technischen Anwendungen häufig benötigt werden, wie beispielsweise die trigonometrischen (Sinus, Kosinus usw.) und hyperbolischen (Hyperbelsinus, Hyperbelkosinus usw.) Funktionen. Beide Arten von Einheiten eignen sich gut zur Umsetzung in hochintegrierte (*VLSI*-)Schaltungen.

Die vorliegende Arbeit führt einen Binärbaum-Ansatz zur Erkennung der führenden von Null verschiedenen Binärziffer (*Leading-One-Detection*, LOD) und partiell lineare Methoden zur Ganzzahlmultiplikation ein, um die Rechenleistung der Standard- und Nichtstandard-Operationen zu verbessern. Es wird gezeigt, dass die weitere Vereinfachung beider Arten von Operationen zwecks leichterer Umsetzung in hochintegrierte (*VLSI*-)Schaltungen möglich ist. Sowohl Durchsatz als auch Latenz der Standard- und Nichtstandard-Operationen wird durch diese Maßnahmen verbessert.

Zur Implementierung der fortgeschrittenen Funktionen kommt der CORDIC-Algorithmus zum Einsatz. Die Herausforderung besteht dabei darin, die Latenz des Algorithmus' zu verringern und seine numerische Genauigkeit zu verbessern. Aus diesem Grund werden zwei Weiterentwicklungen des CORDIC-Grundalgorithmus' betrachtet, nämlich die Doppel- und die Dreifachrotation (*double-rotation* bzw. *triple-rotation*). Die Rechenleistung und Genauigkeit beider Varianten wurde analysiert und mit den Ergebnissen des konventionellen CORDIC-Algorithmus' verglichen; dazu wurden Simulationen in MATLAB durchgeführt. Die eingeführten Weiterentwicklungen des CORDIC-Algorithmus' bieten eine bessere Genauigkeit als der konventionelle Algorithmus; bei gleicher Genauigkeit erfordern sie eine geringere Anzahl von Iterationen und bieten somit eine geringere Latenz. Die verschiedenen Varianten des CORDIC-Algorithmus werden hinsichtlich ihrer effizienten Umsetzung in eine *Pipeline*-Architektur verglichen.

Darauf aufbauend wird eine hochpräzise CORDIC-Funktionseinheit entwickelt, welche zwei Rechenmodi (normale und hohe Genauigkeit) bietet und dafür die Doppel- bzw. die Dreifachrotation nutzt. Diese Funktionseinheit basiert auf einer Festkomma-Zahlendarstellung; daher werden Hilfsfunktionen zur Konvertierung zwischen Gleitkomma- und Festkommadarstellung (*Float-to-Fixed* und *Fixed-to-Float*) eingeführt. Das Zusammenspiel zwischen Gleitkomma-Standard- und -Nichtstandard-Operationen und der Festkomma-CORDIC-Funktionseinheit wird demonstriert, indem diese Einheiten sowohl in einen arithmetischen Koprozessor zur Beschleunigung wissenschaftlicher Rechnungen als auch in einen anwendungsspezifischen Prozessor zur Verarbeitung von Streaming-Daten integriert werden.

Abschließend wird zur Verifikation der vorgestellten CORDIC-Algorithmen ein Phasendetektor für die Strahlphasenregelung eines Schwerionensynchrotrons vorgestellt. In einem Schwerionensynchrotron zirkulieren Teilchenpakete, so genannte *Bunches*. Unter gewissen Umständen kann es zu kohärenten Schwingungen der einzelnen Teilchen innerhalb eines *Bunches* kommen. Verschiedene Schwingungsmoden können dabei auftreten. Diese Schwingungen sind unerwünscht, weswegen eine Strahlphasenregelung eingesetzt wird, um diese Schwingungen zu dämpfen. Dabei wird zunächst nur der einfachste Mode betrachtet, bei dem alle *Bunches* gleichphasig schwingen und sich die *Bunch*-Form nicht verändert. Der Phasendetektor misst die Phasendifferenz zwischen zwei hochfrequenten Signalen, dem Strahlstrom und der Beschleunigungsspannung. Die Strahlphasenregelung ist bestrebt, Schwankungen dieser Phasendifferenz zu dämpfen. Der Phasendetektor wurde in der Hardware-Beschreibungssprache *VHDL* modelliert und mit ModelSim simuliert. Als Stimuli (Beschleunigungsspannung und Strahlstrom) der Simulation wurden sowohl

Simulationsergebnisse eines abstrakten Modells eines Schwerionen-Synchrotrons als auch Messdaten von Maschinenexperimenten am SIS18 des GSI Helmholtzzentrum für Schwerionenforschung verwendet. Die Ausgangssignale des *VHDL*-Modells werden mit anderen, mit MATLAB durchgeführten Simulationen verglichen und so der CORDIC-Algorithmus verifiziert.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CAD	:	Computer Aided Design
LDS	:	Latch D-FF Selector cell
LTI	:	Linear Time Invariant
MAC	:	Multiply-Accumulation
LOD	:	Leading One Detection
CORDIC	:	Coordinate Rotation Digital Computer
MAF	:	Multiplication-Add Fused
PoS	:	Product-of-Sum
SoP	:	Sum-of-Product
BT-Cell	:	Binary-Tree Cell
CRA	:	Carry-Ripple-Adder
LZA	:	Leading-Zero-Anticipator
FLOPS	:	Floating-Point Operation per Second
FR	:	Fetch Instruction Rate
WR	:	Writeback Rate
Minstr	:	Mega-Instruction
Mrpy	:	Mega-Reply
SD	:	Signed-Digit
MAPE	:	Mean Absolute Percent Error
BEP	:	Break-Even-Point
LNS	:	Logarithmic Number System
LUT	:	Look-Up-Table
RMux	:	Multiplexer-based Right Shift
LMux	:	Multiplexer-based Left Shift
DSP	:	Digital Signal Processing

GST	:	Generalized Svoboda and Tung
FPU	:	Floating-Point Unit
FFT	:	Fast Fourier Transform
SDA	:	Sign-Digit-Adder
CSA	:	Carry-Save-Adder
DR	:	Double-Rotation
TR	:	Triple-Rotation
CV	:	Conventional
SIGN-SEL	:	Redundant Sign Selection
SIGN-SEL-NON	:	Non-Redundant Sign Selection
SHR	:	Right Shifter
QI	:	Integer-Bit Part
QF	:	Fractional-Bit Part
MSB	:	Most Significant Bit
SREG	:	Shifter Register
CCU	:	Central Controller Unit
WM Ctrl	:	Write-Memory Control
FPU Ctrl	:	Floating-Point Unit Control
IOB Ctrl	:	Input-Output Control
CNT Ctrl	:	Streaming Counting Control
OCNI	:	On-Chip Network Interface
NoC	:	Network-On-Chip
High-ACC-CORDIC	:	High-Accuracy-CORDIC
FPGA	:	Field Programmable Gate Array

List of Symbols

$\|\|$	:	Concatenation operation
op_1	:	Input operand 1
op_2	:	Input operand 2
op_3	:	Input operand 3
A_s	:	Sign of the number represented by the binary word A
A_e	:	Exponent of the number represented by the binary word A
A_m	:	Mantissa of the number represented by the binary word A
B_s	:	Sign of the number represented by the binary word B
B_e	:	Exponent of the number represented by the binary word B
B_m	:	Mantissa of the number represented by the binary word B
C_s	:	Sign of the number represented by the binary word C
C_e	:	Exponent of the number represented by the binary word C
C_m	:	Mantissa of the number represented by the binary word C
$Shift_{length}$	:	Different value of two exponent values
$g_{A>B}$	:	Boolean value when the binary word A is greater than the binary word B
$g_{A=B}$	:	Boolean value when the binary word A is equal to the binary word B
ϵ	:	Error
$\bar{\epsilon}$	:	Average error
I_d	:	Index's instruction
P_{id}	:	Processor ID
$data - in/out$	:	Input/output data
$valid - in/out$	:	Input/output valid signal
$ack - in/out$	:	Input/output acknowledge signal
$ready - i/osignal$	:	Input/output ready signal
$\#F1$	:	Short instruction format
$\#F2$	:	Long instruction format
$\#R1$	:	Reply format
x_i	:	Value of variable x at the *i-th* iteration

y_i	:	Value of variable y at the *i-th* iteration
z_i	:	Value of variable z at the *i-th* iteration
$Min.\|Error\|$	:	Minimum absolute error
$Max.\|Error\|$	:	Maximum absolute error
$Ave.\|Error\|$	:	Average absolute error
$Std.\ Dev\|Error\|$	:	Standard deviation absolute error
ΔE	:	Different expected error
Δe	:	Different actual error
δ	:	Rotation direction which is either -1 or 1
K_c	:	Constant scaling factor of the conventional CORDIC
K_d	:	Constant scaling factor of the double-rotation CORDIC
K_t	:	Constant scaling factor of the triple-rotation CORDIC
Θ	:	Input angle
β	:	Rotation direction of the redundant CORDIC
$A_{double-rotation}$	:	Area complexity of micro-rotation of the double-rotation CORDIC
$A_{triple-rotation}$	:	Area complexity of micro-rotation of the triple-rotation CORDIC
$rmode$	:	CORDIC's rotation mode
$func$	:	Elementary functions performed by the CORDIC
hs	:	High accuracy mode
T_{ext}	:	Timing complexity of the convergence extension module
T_{pre}	:	Timing complexity of the pre-processing module
T_{dr}	:	Timing complexity of the double-rotation CORDIC module
T_{tr}	:	Timing complexity of the triple-rotation CORDIC module
$T_{micro-dr}$	:	Timing complexity of the micro-rotation of the double-rotation CORDIC module
$T_{micro-tr}$	:	Timing complexity of the micro-rotation of the triple-rotation CORDIC module
T_{post}	:	Timing complexity of the post-processing module
$N_{iter-dr}$	:	Number of iterations of the double-rotation CORDIC
$N_{iter-tr}$	:	Number of iterations of the triple-rotation CORDIC
$Y_{fixed-signed}$	:	Fixed-point representation in sign-magnitude format
$Y_{fixed-2CMP}$	:	Fixed-point representation in 2's complement format
CMD	:	Command
$\Delta\omega$	:	Different phase
2θ	:	Duplicating the micro-rotation angle
3θ	:	Triplicating the micro-rotation angle

List of Units

Frequency	:	hertz	:	Hz
Time	:	second	:	s
FPGA logic cell	:	slice	:	slice
Area/resource on silicon	:	square micro metre	:	μm^2
Computational latency	:	clock cycle	:	#Clock
The number of pipeline	:	stage	:	stage

Chapter 1

Introduction and Overview

Contents

1.1 Background

Today, with increasing computer power, the throughput time for calculation and analysis of complex data sets is decreasing. Computers can calculate, perform, and analyse huge and complex data within a short time. For example in the field of scientific research in nuclear physics, the building blocks and interactions of atomic nuclei must be analysed in order to build, e.g. nuclear power plants. However, nuclear physics is also used in many other research areas such as with medicine, magnetic resonance imaging, ion synchrotron [59], etc. Another scientific area that requires computers for computation and analysis is crystallography. Crystallography considers the geometric forms of crystals. Crystallography is the process that identifies how to describe, classify and measure crystals, revealing what forces made them and what activities occur within them. It is essential for producing materials like metals and alloys, ceramics, glasses, polymers, and medicines.

In engineering, computers are employed for design, simulation and optimization before actual engineering implementation. Model or design simulations explore new insights into innovative analysis, leading to improved technology. For instance, civil and mechanical engineering uses *Computer-Aided-Design* (CAD) for architectural and structural drawing and investigation. *CAD* involves the use of computer technology for the process of design and simulation, where the steps of drafting, documentation, and man-

ufacturing are described and synthesized by computers. The tool is important for industrial design which is extensively applied for many applications, including in the automotive, shipbuilding, structural, architectural, and aerospace industries. An example of where it is employed is in the broadcasting industry when simulating signal processing and conversion. *CAD* has become a major tool for electrical and electronic research in computational geometry, computer graphic and discrete differential geometry.

Due to heavy processing requirements, the performance and efficiency of computers have been improved and continuously developed in order to meet high computational accuracy. Two major elements when considering a computers performance are the speed and accuracy of computations, both elements rely on hardware and software components [12]. For example, parallel computer architecture, shared-memory, message-passing, data-parallel, and data-driven computing architectures are discussed in the hardware approach [36]. In the software approach, a parallel programming complier or high performance computer architecture are examined [46]. Moreover, the efficiency of hardware and software synchronization also significantly improves the performance and efficiency of computers. Normally, software is designed based on targeted hardware components. If a hardware platform is easy to use and is highly effective, then its software complexity will be low [40] [35] [22]. Consequently, the enhancement of hardware components becomes a major task when driving the improvement and development of computer technology to support the demands of computational applications.

Several pieces of literature examine the improvement of hardware components in terms of computational speed and accuracy. For example, high speed memory architectures for specific applications were introduced by Y. Oshima et al. [89], S. H. Kang et al. [56], R. Pinkham et al. [95], R. Hashemian [45], where they customized the data storage for the purpose of improving storage capability and accessibility corresponding to their applications demands. Performance degradation problems of long-latency memory access was alleviated by cache memory. F. Dahlgren et al. [25] introduced cache-only memory architecture which increased the chance of data being locally available. It was employed in the multiprocessor platform, where a shared-memory paradigm was applied for parallel programming applications in order to reduce the impact of frequent long-latency memory access. A. Meixner et al. [76] proposed cache-coherent to execute critical network services and database management systems for multi-threaded computer servers.

An arbitrary component becomes another type of hardware component that can improve the performance and efficiency of computers. It provided not only an interconnection among units but also control functions which are applied to manipulate data communication. S. Radhakrishnan et al. [99] introduced Intel 5000 chipset architecture to significantly improve the performance of the Intel Xeon multiprocessor core for computation-intensive applications such as flight simulators, computational fluid dynamics, finite-element analysis, etc. K. Wang et al. [127] proposed the MPC105 PCI bridge/memory controller to support the PowerPC 601, 603, and 640 microprocessors. The bride chip integrated peripheral devices on the PCI bus, a secondary cache controller, and the high-

performance memory controller that supports the DRAM or SRAM and ROM or flash ROM.

Several previous works discuss the performance, efficiency, and accuracy of co-processor/processors. T. Sasaki et al. [104] proposed a design method in the pipeline technique dealing with low-energy and high-performance computing processors for mobile and portable devices. They applied a *Latch D-FF Selector cell* (*LDS*), where it can be reconfigured to be either a *D-Flip-Flop* (*DFF*) or a latch in a computational stage of a processor at runtime. The *DFF* and the latch were used in high-speed mode and low-energy mode. A high-performance processor specific for embedded real-time control applications particularly in vehicles was introduced by R. Cumplido et al. [24]. The processor was designed based on a stage-machine mechanism to support the *Linear Time Invariant* (*LTI*) control, where the *Multiply-Accumulation* (*MAC*) function is the main function in its arithmetic unit. Variable-precision arithmetic processors for scientific applications were proposed by M. J. Schulte et al. [107]. These processors allowed the programmer to specify the precision of the computation, determine the accuracy of the results, and recomputed inaccurate results with higher precision. Since accuracy and reliability became a main consideration of this processor, iterative architecture based on floating-point representation was applied in the design and implementation. A. M. Psomoulis et al. [97] introduced specific processor architecture for aerospace imaging instruments. The architecture offered vibration tolerant, thermal, radiation, high performance, high reliability, high accuracy, and intelligence for computation.

To summarize, many efforts have been made to explore different designs and architectures of processors for both hardware and software, particularly with arithmetic units. Since there are few arithmetic units dedicated on scientific and engineering computer targeting of performance, efficiency and accuracy, this thesis considers an optimal arithmetic algorithm, design, architecture, and analysis of an arithmetic unit specific for scientific and engineering applications.

1.2 Motivations

The idea and motivation applied for the arithmetic algorithm, design and analysis of the arithmetic unit are as follows.

- *High performance, high efficiency, and high computational accuracy of the arithmetic unit of a scientific co-processor/processor*: The design and architecture of the arithmetic unit of a co-processor/processor specific for a scientific application mainly focuses on performance. Therefore, an idea with regards to the design and structural development of the arithmetic unit based on computational accuracy and latency is proposed.

- *Low cost and per-formability of elementary functions essential for scientific and engineering applications*: Since the elementary functions such as sine, cosine, hyperbolic sine,

hyperbolic cosine, etc., are frequently employed in scientific and engineering applications, these functions should be efficiently created with a low algorithmic complexity, low computational latency, and a high computational accuracy.

- *Architectural simplicity for verification, VLSI implementation, and integration*: The *VLSI* architecture of the arithmetic unit which can perform both basic operations and elementary functions in floating-point and fixed-point formats should provide a low complexity and ease of verification, implementation and usage.

1.3 Research Objectives and Scope

As this dissertation proposes the improvement of performance, efficiency, and computational accuracy of a floating-point and fixed-point arithmetic unit to support the scientific and engineering applications, the research scope is an algorithm optimization, verification, and design for *VLSI* implementation of the arithmetic unit. Some issues and aspects of the efficient computational algorithm optimisation, verification method, and modularity architecture of standard and non-standard operators and elementary function operators are discussed.

The general objective of the doctoral thesis is to present algorithms, investigation methods, and design concepts and generic architecture of the floating-point and fixed-point arithmetic unit. The specific objectives are:

- to present algorithms for floating-point standard and non-standard operations, where the algorithms are easy for verification, investigation, and *VLSI* implementation as well as to introduce a method of improving the performance of a Leading-One-Detection unit and an integer multiplier unit [138], [142],
- to present extension-rotation CORDIC methods in order to alleviate the long computational latency and to introduce a verification method for examining the computational accuracy of the proposed CORDIC methods [144],
- to design a high precision CORDIC algorithm, and to consider a hardware investigation and evaluation of the CORDICs micro-rotation which can be applied for design and VLSI implementation [139], [137],
- to introduce exemplars of design and architecture of the floating-point arithmetic unit for an intensive-computation accelerator/processor [143].

Therefore, the main scope of this thesis is

1) Floating-point standard and non-standard algorithms,

2) Computational accuracy analysis,

3) CORDIC algorithm in double-rotation and triple-rotation methods,

4) Convergence range extension,

5) A high precision CORDIC algorithm,

6) Data conversion algorithm,

7) Combinational architecture of the floating-point and fixed-point arithmetic units for *VLSI* implementation of an accelerator/processor.

1.4 Thesis Outline

- *Chapter 2*: This chapter describes the improvement of basic floating-point operators, i.e. an adder/subtractor, a multiplier, a product-of-sum operator, and a sum-of-product operator. Their optimal algorithms that are suitable for design and *VLSI* implementation of the basic floating-point operators are introduced. The algorithms can be applied for single- and double-precision IEEE standard floating-point representation. Common functions, i.e. right/left shifting and *LOD*, and highly critical delays affecting the performance of the floating-point operators will be investigated. A multiplexer-based shifting technique and a binary-tree searching technique are applied to minimize the critical delay of the floating-point operators. In addition, an integer multiplier which is a common integer operator for a floating-point multiplier, a product-of-sum and a sum-of-product is enhanced by a linear partial method in order to enhance performance. Finally, they are applied to the design of a floating-point accelerator which can be used to increase the computational performance of general-purpose processors.

- *Chapter 3*: This chapter introduces the rotation-extension CORDIC methods, i.e. double-rotation and triple-rotation, for the objective of improving the performance, time efficiency, and computational accuracy of the CORDIC algorithm in radix-2. In the double-rotation and triple-rotation methods, the convergences of the CORDIC are accelerated by duplicating and triplicating the micro-rotation angles to be 2θ and 3θ, respectively. Convergence range and computational accuracy of elementary functions performed by using the CORDIC methods in rotation mode and vectoring mode on the circular, hyperbolic, and linear coordinate systems are examined, investigated and compared to Matlab/Simulink ideal results. The comparison results show that the proposed CORDIC methods provide higher accuracy than the conventional CORDIC at the same number of iterations. Moreover, extension functions derived from CORDICs elementary functions in hyperbolic coordinate systems, i.e. natural logarithm and square root, are considered and investigated. Finally, convergence range problems of the CORDIC are discussed.

- *Chapter 4*: This chapter discusses the design and architecture of an arithmetic unit which supports both floating-point and fixed-point computation. The unit can perform basic mathematical functions which are necessary for scientific calculation, i.e. standard functions, non-standard functions, and elementary functions. The design and architecture of the proposed CORDIC methods will be considered and analysed. The design and architecture of an accelerator and a reconfigurable streaming processor are proposed, where the accelerator can be applied to cooperate with a main processor in order to sustain floating-point computation. The reconfigurable streaming processor is designed for a specific application which is used when processing streamed data.

- *Chapter 5*: This chapter presents an illustration of the proposed CORDIC in a circular coordinate system on vectoring to perform the beam phase and magnitude detector employed in the closed-loop control system for heavy ion synchrotron application. An overview of the closed-loop control system is described; afterwards the algorithm, design, implementation, and simulation of the digital phase and magnitude detection module will be discussed. The design of the digital phase and magnitude detector is modelled and simulated based on VHDL. The simulation results based on the actual digital signals of the closed-loop control are compared with Matlab/Simulink in order to verify the proposed CORDICs computation.

- *Chapter 6*: The new contributions of this thesis are summarized in this chapter. Any directions for future works will also be briefly described in this chapter.

Chapter 2

Improvement of Standard and Non-Standard Floating-Point Operations

Contents

The improvement of floating-point operators which are widely employed in digital signal processing application areas is described in this chapter. The basic floating-point operators, i.e. the standard operators and non-standard operators with their simple algorithms suitable for design and *VLSI* implementation are introduced. The algorithms can be applied for implementation of the floating-point operations in single- and double-precision IEEE standard floating-point representations. For the sake of simplicity, the 32-bit single-precision IEEE standard floating-point format is examined. From architectural investigation, common functions, i.e. right/left shifting and leading-one-detection (*LOD*), present high critical delays effecting the performance of the floating-point operators. To minimize the critical delays, a multiplexer-based shifting technique and a binary-tree searching technique are applied. Moreover, an integer multiplier which is a common integer operator for a floating-point multiplier, a product-of-sum and a sum-of-product is improved by a linear partial method in order to reduce critical delays. The standard and non-standard floating-point operators are synthesized on the Xilinx Virtex5 xc5vlx110t-3ff-1136 FPGA technology and the 130-nm silicon technology targeting at frequencies of 200 MHz and 1 GHz respectively. Finally, they are utilized for the design of a floating-point accelerator which can be used for increasing the computational performance of general-purpose processors such as open cores Motorola MC6820 and LeonII, where floating-point execution units are non-integrated.

2.1 State-of-the-Art

Requirements for real-time highly accurate computations have considerably increased in recent applications. Critical applications, like medical image processing [37] or linear phase FIR digital filters [15], rely on floating-point computations for accurate and efficient processing. The majority of modern processors such as Motorola 6840 integrate a hardware floating-point arithmetic unit in order to fulfil the computational accuracy demands whereas classic processors perform floating-point arithmetic functions using software libraries. Although the operations can be introduced by the software method, the computation is very slow in comparison to hardware implement. Several strategies for the implementation of floating-point units, accelerators, and processors were reported in related works in the following areas.

2.1.1 Chip Design and Functionality

In 1983, Huntsman et al. [52] introduced the MC68881 floating-point co-processor used to cooperate with Motorola's M68000 32-bit processor family. The MIPS R3010 chip [101] specified for the R3000 RISC processor was proposed in order to reduce design cost. It provides the basic floating-point operations, i.e. addition/subtraction, multiplication,

and division. Maurer [73] introduced the WE32106 math accelerator, but it mainly focused on verification techniques. Nakayama et al. [84] designed an 80-bit floating-point co-processor providing 24 instructions and 22 mathematic functions, where the adder/-subtractor and multiplier were designed in pipeline structure, but the divider performed using the CORDIC algorithm, which provides high computational latency. Kawasaki et al. [58] introduced a pipeline floating-point co-processor cooperating with the *GMICROs* processor as an intelligent CPU for the *TRON* architecture. The co-processor has 23 instructions to build basic and trigonometric operations.

2.1.2 Improvement of Performance and Efficiency at Runtime

Darley et al. [26] proposed the TMS390C602A floating-point co-processor to cooperate with the SPARC TMS390C601 integer processor. They optimized the system performance by balancing the floating-point execution throughput and instruction fetching. This method demonstrated higher performance while dramatically cutting system costs. A 16-bit pipelining floating-point co-processor on FPGA was investigated by Fritz and Valerij [74]. Based on the SIMD structure, the co-processor is placed between the processor and the main memory. When the processor needs to execute a floating-point operation, the processor will simultaneously send an instruction to the co-processor and the address of the given operands to the memory. The co-processor can thus directly fetch the operands from the memory.

2.1.3 Enhancement of Designs and Algorithms of Basic Arithmetic Units

Nielsen et al. [87] proposed a pipelined floating-point addition algorithm with 4-stages in packet forwarding format which was a redundant representation of the floating-point number in order to improve the mantissa fraction. Chen et al. [20] introduced the architecture of a multiplication-add fused (*MAF*) unit to reduce the three-word-length addition to two-word-length to carry propagation in a conventional *MAF*. Either leading-one/zero-detection or-prediction common functions for floating-point operations were considered by Javier et al. [16], Suzuki et al. [113], Hokenek et al. [48], and Schmookler et al. [105]. From the above literature the performance of the floating-point operators can be improved by considering design and architecture of integer adder, integer multiplier and leading-one/zero-detection or-prediction which will be considered in this chapter.

In real-time computations such as digital filter applications [140], time constraint is a main factor for design consideration, where the filter's calculation has to be finished before a new sample arrives. If the floating-point computation units are performed by using software libraries on a process, which obviously provides longer latency than hardware, the targeting time constraint can not be achieved. Clearly, modern processors, where the floating-point units are embedded can fulfil the requirement. Classic processors can also support the constraint by redesign, but its cost and complexity become a major problem

for consideration. In floating-point arithmetic units, the loss of their performance comes from critical delays on common functions such as leading-one-detection, shifting functions and integer multiplier. To reduce these delays, the common functions have to be investigated and improved. In modern digital applications, multi-processor system platforms are widely used due to their acceleration ability on an application's computation. Normally, the processors execute their floating-point tasks using their own floating-point library which consume more resources and time. Thus, a hardware-sharing concept, where one floating-point accelerator is shared by multi-processors will not only reduce the consumed resources, but also computational time and power consumption.

The remainder of this chapter deals with

1) Floating-point algorithms and the standard and non-standard operators,

2) Design and enhancement of the leading-one/zero-detection and right/left shifting functions as well as a partial liner method for an integer multiplier,

3) Implementation and investigation of floating-point operators, the design and architecture of a floating-point arithmetic accelerator.

2.2 Floating-Point Operation Algorithm and Analysis

The algorithms of standard floating-point operators, adder/subtractor and multiplier, and non-standard floating-point operators, product-of-sum (*PoS*) operator and sum-of-product (*SoP*) are analysed and considered to increase computation performance. The algorithms can be applied for the single-and double-precision IEEE standard floating-point representations [53] as shown in Fig. 2.1. The single- and double-precision IEEE standard floating-point formats are binary computing formats that occupies 4 bytes (32 bits) and 8 bytes (64 bits). Both floating-point IEEE standard formants comprise three basic components, i.e. sign, exponent, and mantissa. The mantissa is composed of fraction and implicit leading digit. Tab. 2.1 shows the layout of the single- and double-precision IEEE standard floating-point formats, where a number of bits of each field are presented in square brackets.

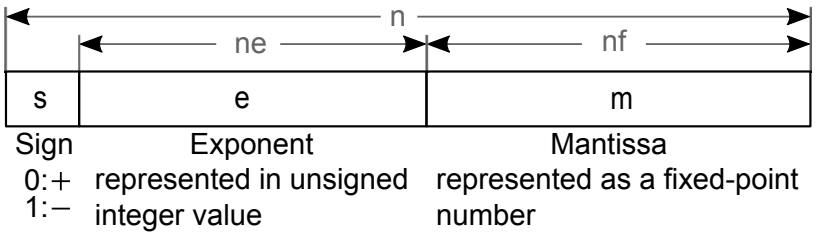


Fig. 2.1: IEEE standard floating-point format

Tab. 2.1: The layout of the single- and double-precision IEEE standards floating-point representations.

IEEE standard	*n*	Sign	Exponent (*ne*)	Mantissa (*nf*)
Single-precision	32	1[31]	8[30:23]	23[22:0]
Double-precision	64	1[63]	11[62:52]	52[51:0]

2.2.1 Common Functions

The common functions, i.e. *Unpacking* function, *Comparison* function, and *Norm* functions which are employed to perform the floating-point operational algorithms are examined. For the sake of simplification, all examples in this chapter are introduced in the single-precision IEEE standard representation, where *n=32*, *ne=8*, and *nf=23*.

2.2.1.1 Unpacking function

This function as shown in Algorithm 1 extracts two input operands, op_1 and op_2, into the two group of the Sign, Exponent, and Mantissa fraction. A_s, A_e, and A_m are in group A whereas B_s, B_e, and B_m are in group B. The carry-bit and guard-bit, $b'01$, are padded on the most significant bit (*MSB*) of the mantissa fraction, where $||$ is a concatenation operation.

Algorithm 1 Unpacking(op_1, op_2, n, ne, nf)

1: $A_s = op_1(n-1)$, $A_e = op_1(n-2:n-ne-2)$;
2: $A_m = b'01||op_1(nf-1:0)$;
3: $B_s = op_2(n-1)$, $B_e = op_2(n-2:n-ne-2)$;
4: $B_m = b'01||op_2(nf-1:0)$;
5: **return** $A_s, A_e, A_m, B_s, B_e, B_m$

For example, assume that $op_1 = -100(h'C2C80000)$, $op_2 = 200(h'43480000)$, $n = 32$, $ne = 8$, $nf = 23$. After executing the *Unpacking* function, the output results will be $A_s = b'1$, $A_e = b'10000101$, $A_m = b'01100100000000000000000000$, $B_s = b'0$, $B_e = b'10000110$, $B_m = b'01100100000000000000000000$.

2.2.1.2 Comparison function

The function compares two input operands fractioned into group A and group B. Normally, the comparison can be done by *If-Statement*, where the two signs, A_s and B_s, are first compared, and then the two exponents, A_e and B_e, and mantissas, A_m and B_m, will be compared respectively. By means of this method, a long critical delay will appear. In order to minimize the delay, parallel comparison based on combinational circuit is introduced. Tab. 2.2 shows a possible case of the operand A and B in truth-table, where p, q, and z depend on A_e, B_e, A_m, and B_m.

Tab. 2.2: Representing the relationship of the fraction of the operands *A* and *B* corresponding to g_e and g_m in truth-table.

case	g_e	g_m	$g_{A>B}$	$g_{A=B}$
1	p	p	$b'1$	$b'0$
2	p	q	$b'1$	$b'0$
3	p	z	$b'1$	$b'0$
4	q	p	$b'1$	$b'0$
5	q	q	$b'0$	$b'1$
6	q	z	$b'0$	$b'0$
7	z	p	$b'0$	$b'0$
8	z	q	$b'0$	$b'0$
9	z	z	$b'0$	$b'0$

$p = b'01$, $q = b'11$, $z = b'00$

For instance, assume that g_e and g_m are greatening results of exponent and mantissa values of two input operands. p, q, and z are defined as $p = b'01$, $q = b'11$, and $z = b'00$. The 2^{nd} case, where $g_e = p$ and $g_m = q$ means that A_e is greater than B_e and A_m, is the same as B_m. The two outputs, $g_{A>B}$ and $g_{A=B}$, are respectively set to $b'1$ and $b'0$, where they cover the condition that the operand A is greater than the operand B. The 5^{th} case shows the condition that the operand A is equal to the operand B with $g_e = q$ and $g_m = q$, the two outputs are set to $b'0$ and $b'1$. Algorithm 2 presents the comparison function derived from Tab. 2.2.

Form the previous computational results where $A_s = b'1$, $A_e = b'10000101$, $A_m = b'01100100000000000000000$, $B_s = b'0$, $B_e = b'10000110$, $B_m = b'01100100000000000000000$, after the *Comparison* function is executed, the internal variables g_e and g_m will be $b'00$ and $b'11$. By executing the combinational logic in Lines 15 and 16, the output results $g_{A=B}$ and $g_{A>B}$ will equal to $b'0$ and $b'0$ as the 8^{th} case on Tab. 2.2.

2.2.1.3 Norm function

The sign (*Sign*), exponent (*E*), and mantissa (*M*) are normalized to conform to the IEEE standard format. The mantissa *M* is shifted to the most significant bit (*MSB*) depending on a given *Position* variable. Whereas the exponent *E* will be either added or subtracted by one, the *Position* corresponds to the carry-bit and guard-bit of the mantissa *M*. The sign, the adapted exponent, and the mantissa are finally packed together. The *Norm* function is illustrated in Algorithm 3.

Algorithm 2 Comparison(A_e,B_e,A_m,B_m)

if $(A_e > B_e)$ **then**
    $g_e = b'01$;
else if $A_e = B_e$ **then**
    $g_e = b'11$;
else
    $g_e = b'00$;
end if
if $(A_m > B_m)$ **then**
    $g_m = b'01$;
else if $(A_m = B_m)$ **then**
    $g_m = b'11$;
else
    $g_m = b'00$;
end if
$g_{A>B} = ((\sim g_e(1)) \cdot g_e(0)) \cdot ((\sim g_m(1)) + g_m(0))) + (g_e(0) \cdot (\sim g_m(1)) \cdot g_m(0))$;
$g_{A=B} = g_e(1) \cdot g_e(0) \cdot g_m(1) \cdot g_m(0)$;
return $g_{A>B}$, $g_{A=B}$

Algorithm 3 Norm(*Sign, E, M, Position, n, ne, nf*)

$X(n-1) = Sign$;
if $(M(nf+1:nf) = b'10) or (M(nf+1:nf) = b'11)$ **then**
    $X(n-2:n-ne-1) = E+1$
    $X(nf-1:0) = M(nf:1)$
else
    $X(n-2:n-ne-1) = E - Position$
    $X(nf-1:0) = Shift(M, Position, Left)$
end if
return X

2.2.1.4 Unpacking3 function

This is a common function specially for non-standard floating-point operators. Like the *Unpacking* function, the function has 3-input operands, op_1, op_2, and op_3 as shown in Algorithm 4. The outputs are performed in three groups of sign, exponent, and mantissa fractions, A_s, A_e, A_m, B_s, B_e, B_m, C_s, C_e, and C_m respectively.

Algorithm 4 Unpacking3(op_1,op_2,op_3, n, ne, nf)

1: $A_s = op_1(n-1)$, $A_e = op_1(n-2:n-ne-2)$;
2: $A_m = b'01||op_1(nf-1:0)$;
3: $B_s = op_2(n-1)$, $B_e = op_2(n-2:n-ne-2)$;
4: $B_m = b'01||op_2(nf-1:0)$;
5: $C_s = op_3(n-1)$, $C_e = op_3(n-2:n-ne-2)$;
6: $C_m = b'01||op_3(nf-1:0)$;
7: **return** A_s, A_e, A_m, B_s, B_e, B_m, C_s, C_e, C_m

2.2.2 Standard Operation

2.2.2.1 Floating-Point Addition/Subtraction

The floating-point addition/subtraction algorithm, detailed by Algorithm 5, comprises the common functions which have been introduced in section 2.2.1. The algorithm is built with respect to design simplicity and implementation in digital hardware.

Tab. 2.3: Relationship of the enabled sign bit $(Sign)$ of the operands A and B corresponding to A_s, B_s, $g_{A>B}$, and $g_{A=B}$.

case	A_s	B_s	$g_{A>B}$	$g_{A=B}$	$Sign$
5	$b'0$	$b'1$	$b'0$	$b'0$	$b'1$
10	$b'1$	$b'0$	$b'1$	$b'0$	$b'1$
13	$b'1$	$b'1$	$b'0$	$b'0$	$b'1$
14	$b'1$	$b'1$	$b'0$	$b'1$	$b'1$
15	$b'1$	$b'1$	$b'1$	$b'0$	$b'1$
16	$b'1$	$b'1$	$b'1$	$b'1$	$b'1$

The proposed algorithm has been split in five steps for both addition and subtraction as described below:

- **Step 1** Unpacking: The sign bits of the two operands, op_1 and op_2, are first evaluated by an XOR operation with the sub variable. Afterwards, both operands will be fractioned into 3 main triples, sign, exponent, and mantissa, by the *Unpacking* function which outputs the variables A_s, A_e, A_m, B_s, B_e, and B_m, respectively.

Algorithm 5 Floating-point adder/subtracter (*FP-Adder*)

Require: op_1, op_2, n, ne, nf, sub

{ *Step 1* : Unpacking}

$op_2(n-1) = sub \oplus op_2(n-1)$

[A_s, A_e, A_m, B_s, B_e, B_m]=$Unpacking(op_1, op_2, n, ne, nf)$

{ *Step 2* : Comparing and sing evaluation}

[$g_{A>B}$, $g_{A=B}$]= $Comparison(A_e, B_e, A_m, B_m)$

$Sign = (A_s \cdot B_s) + (A_s \cdot g_{A>B} \cdot (\sim g_{A=B})) + (B_s \cdot (\sim g_{A>B}) \cdot (\sim g_{A=B}))$

{ *Step 3* : Exponent subtraction, mantissa swap, and shift}

if $(g_{A>B} = b'1) or (g_{A=B} = b'1)$ **then**

    $E_{add} = A_e$, $M_{l1} = A_m$, $M_{l2} = B_m$

    $Shift_{length} = A_e - B_e$

else

    $E_{add} = B_e$, $M_{l1} = B_m$, $M_{l2} = A_m$

    $Shift_{length} = B_e - A_e$

end if

$M_{adp} = Shift(M_{l2}, Shift_{length}, Right)$

{ *Step 4* : Mantissa addition/subtraction}

if $((A_s \oplus B_s) = b'0)$ **then**

    $M_{add} = M_{l1} + M_{adp}$

else

    $M_{add} = M_{l1} - M_{adp}$

end if

{ *Step 5* : Leading-One-Detection and normalization}

$Position = LOD(M_{add})$

$X = Norm(Sign, E_{add}, M_{add}, Position, n, ne, nf)$

- **Step 2** Comparison and sign evaluation: The partial exponents and mantissas, A_e, A_m, B_e, and B_m, are compared by the *Comparison* function to determine the greatest value between op_1 and op_2. Then, the $Sign$ is evaluated by the optimized combinational logic.

- **Step 3** Exponent subtraction and mantissa swap: The results of the comparison $g_{A>B}$ and $g_{A=B}$ are applied to compute the difference of the two exponents $Shift_{length}$ and to swap the exponents and the mantissas. The $Shift_{length}$ will be applied to adjust the lower mantissa M_{l2} using the shifting function.

- **Step 4** Mantissa addition/subtraction: The addition/subtraction of the mantissas depends on the xoring result of A_s and B_s.

- **Step 5** Leading-One-Detection and normalization: The first bit one of the addition/-subtraction result of the mantissas is searched by the LOD function, where the detected result will be employed to normalize the final exponent and the final mantissa generated from previous steps. The *Norm* function will finally pack them together.

The details of the LOD and shifting functions have been fully described in section 2.3.

2.2.2.2 Floating-Point Multiplication

In comparison to the floating-point addition/subtraction algorithm, the complexity of the floating-point multiplication algorithm is lower as illustrated by Algorithm 6. It is also performed in five steps described below:

- **Step 1** Unpacking: The two operands are fractioned by the *Unpacking* function.

- **Step 2** Subtracting and comparing: The *Comparison* function is applied to compare the exponents and the mantissa, A_e, B_e, A_m, and B_m.

- **Step 3** Swap and Sign evaluation: The $Swap$ and $Sign$ are evaluated using AND and XOR operations. The $Swap$ is then employed to perform swapping of the two exponents and the two mantissas.

- **Step 4** Exponent and mantissa determination: The two exponents, E_{l1} and E_{l2}, are subtracted and added by 127 in the signed integer form in order to output the final exponent E_{mul}. The unsigned integer multiplication is utilized to compute the final mantissa M_{mul}.

- **Step 5** Leading-One-Detection and Normalization: The process is the same as step 5 of the addition/subtraction algorithm.

Algorithm 6 Floating-point multiplier (*FP-Multiplier*)

Require: op_1, op_2, n, ne, nf

{ *Step 1* : unpacking}

1: $[A_s, A_e, A_m, B_s, B_e, B_m] = Unpacking(op_1, op_2, n, ne, nf)$

{ *Step 2* : Subtracting and comparing}

2: $[g_{A>B}, g_{A=B}] = Comparison(A_e, B_e, A_m, B_m)$

{ *Step 3* : Swap and sign evaluation}

3: $Swap = (\sim g_{A>B}) \cdot (\sim g_{A=B})$

4: $Sign = A_s \oplus B_s$

5: **if** $(Swap = b'0)$ **then**

6: $E_{l1} = A_e, E_{l2} = B_e, M_{l1} = A_m, M_{l2} = B_m$

7: **else**

8: $E_{l1} = B_e, E_{l2} = A_e, M_{l1} = B_m, M_{l2} = A_m$

9: **end if**

{ *Step 4* : Exponent and mantissa determination}

10: $E_{mul} = E_{l1} - E_{l2} + 127$

11: $M_{mul} = M_{l1} \times M_{l2}$

{ *Step 5* : Leading-One-Detection and normalization}

12: $Position = LOD(M_{mul})$

13: $X = Norm(Sign, E_{mul}, M_{mul}, Position, n, ne, nf)$

2.2.3 Non-Standard Operation

There are non-standard floating-point arithmetic operations with 3 input operands being widely used in digital signal processing applications. *PoS* and *SoP* operators, $(A+B) \times C$ and $(A \times B) + C$, are frequently employed in multimedia [29] [140] and filtering applications. These operators can be performed using basic floating-point addition and the floating-point multiplication in cascade. However, in order to improve the performance of the floating-point unit, algorithms for the *PoS* and *SoP* operators are introduced by the fusion of the floating-point addition and multiplication algorithms.

2.2.3.1 Floating-Point Product-of-Sum Operation

The floating-point product-of-sum (*FP-PoS*) operator, $(A + B) \times C$, is a combination of the floating-point adder and multiplier. The *FP-PoS* algorithm shown in Algorithm 7 is described below:

- **Step 1** Unpacking : The three operands, op_1, op_2, op_3, are fractioned in A_s, A_e, A_m, B_s, B_e, B_m, C_s, C_e, and C_m by the *Unpacking3* function.
- **Step 2** Comparing and sign evaluation: The two exponents, A_e and B_e, and the two mantissas, A_m and B_m, are sorted by th *Comparison* function. Then, the sign is determined.

Algorithm 7 Floating-point product-of-sum (*FP-PoS*)

Require: op_1, op_2, op_3, n, ne, nf

{**Step 1** : Unpacking}

$[A_s, A_e, A_m, B_s, B_e, B_m, C_s, C_e, C_m] = Unpacking3(op_1, op_2, op_3, n, ne, nf)$

{**Step 2** : Comparing and sign evaluation}

$[g_{A>B}, g_{A=B}] = Comparison(A_e, B_e, A_m, B_m)$

$Sign_l = A_s \cdot B_s + A_s \cdot g_{A>B} \cdot (\sim g_{A=B}) + B_s \cdot (\sim g_{A>B}) \cdot (\sim g_{A=B})$;

{**Step 3** : Exponent subtraction, mantissa swap, and final Sign evaluation}

$Sign_{pos} = Sign_l \oplus C_s$

$Sub_{l1} = A_s \oplus B_s$

if $(g_{A>B} = b'1) or (g_{A=B} = b'1)$ **then**

    $E_{l1} = A_e - B_e$, $E_{l2} = B_e$, $M_{l1} = A_m$, $M_{l2} = B_m$

else

    $E_{l1} = B_e - A_e$, $E_{l2} = A_e$, $M_{l1} = B_m$, $M_{l2} = A_m$

end if

{**Step 4** : Exponent and mantissa determination}

if $(g_{A>B} = b'1)$ **then**

    $E_{add} = E_{l2} - C_e + 127$

else

    $E_{add} = C_e - E_{l2} + 127$

end if

$M_{shift} = b'0 || Shift(M_{l2}, E_{l1}, Right)$

if $(Sub_{l1} = b'0)$ **then**

    $M_{add} = M_{l1} + M_{shift}$

else

    $M_{add} = M_{l1} - M_{shift}$

end if

{**Step 5** : Leading-one-detection, final exponent and mantissa alignment}

$Position = LOD(M_{add})$

if $(M_{add}(nf + 1{:}nf) = b'10) or (M_{add}(nf + 1 : nf) = b'11)$ **then**

    $E_{pos} = E_{l1} + E_{add} + 1$

    $M_{align} = b'01 || M_{add}$

else

    $E_{pos} = E_{l1} + E_{add} - Position$

    $M_{align} = b'01 || Shift(M_{add}, Position, Left)$

end if

{**Step 6** : Final mantissa determination}

$M_{pos} = M_{align} \times C_m$

{ **Step 7** : Leading-one-detection and normalization}

$Position = LOD(M_{pos})$

$X = Norm(Sign_{pos}, E_{pos}, M_{pos}, Position, n, ne, nf)$

- **Step 3** Exponent subtraction, mantissa swap, and final sign evaluation: The final sign $Sign_{pos}$ is evaluated by xoring $Sign_l$ and C_s. Meanwhile, the sign bit Sub_l is xored by A_s and B_s. The comparison results will be used to swap mantissas and to compute the exponent difference.
- **Step 4** Exponent and mantissa determination: The exponents and mantissas are computed by adding and shifting.
- **Step 5** Leading-one-detection (*LOD*), final exponent, and mantissa alignment: The first bit one is searched by the function *LDO*. The final exponent and mantissa alignment corresponding to the additional result of the mantissa of op_1, op_2 are accumulated by adding and shifting.
- **Step 6** Final mantissa determination: The two mantissas are multiplied together using an unsigned integer multiplication.
- **Step 7** Leading-one-detection and normalization: Operating as step 5 of Algorithm 5

2.2.3.2 Floating-Point Sum-of-Product Operation

The floating-point sum-of-product (*FP-SoP*) operation algorithm $((A \times B) + C)$ is detailed below :

- **Step 1** Unpacking: Operating as step 1 of Algorithm 7.
- **Step 2** Comparing, exponent subtraction, mantissa swap, and sign evaluation: The two exponents, A_e and B_e, and the two mantissas, A_m and B_m, are compared by *Comparison* function. The comparison result will be used to swap the mantissas, to compute an intermediate exponent, and to evaluate the sign by xoring A_s and B_s.
- **Step 3** Mantissa multiplication: The two mantissas are multiplied together in form of unsigned integer multiplication.
- **Step 4** Leading-one-detection, exponent, and mantissa alignment: The length of shifting is determined by function *LOD*. The exponents are calculated and the mantissas are aligned by addition and shifting.
- **Step 5** Comparing, final sign evaluation, exponent subtraction, and mantissa swap: The two exponents and the two mantissas are compared together and the final sign is determined by xoring. Afterwards, the intermediate mantissas and the intermediate exponents are swapped.
- **Step 6** Shift and final mantissa determination: The mantissa is aligned and the final mantissa M_{op} is determined.
- **Step 7** Leading-one-detection and normalization: Operating as step 5 of Algorithm 5

Algorithm 8 Floating-point sum-of-product (*FP-SoP*)

Require: $op_1, op_2, op_3, n, ne, nf$

{ *Step 1* : Unpacking}

[$A_s, A_e, A_m, B_s, B_e, B_m, C_s, C_e, C_m$]= Unpacking3($op_1, op_2, op_3, n, ne, nf$)

{ *Step 2* : Comparing, exponent subtraction, mantissa swap, and sign evaluation}

$[g1_{A>B}, g1_{A=B}] = Comparison(A_e, B_e, A_m, B_m)$

if $(g1_{A>B} = b'1) or (g1_{A=B} = b'1)$ **then**

    $M_{l1} = A_m$, $M_{l2} = B_m$, $E_{l2} = B_e - A_e + 127$

else

    $M_{l1} = B_m$, $M_{l2} = A_m$, $E_{l2} = B_e - A_e + 127$

end if

$Sign_{mul} = A_s \oplus B_s$

{ *Step 3* : Mantissa multiplication}

$M_{mul} = M_{l1} \times M_{l2}$

{ *Step 4* : Leading-one-detection, exponent and mantissa alignment}

$Position = LOD(M_{mul})$

if $(M_{mul}(2 \cdot nf + 1 : 2 \cdot nf) = b'11) or (M_{mul}(2 \cdot nf + 1 : 2 \cdot nf) = b'10)$ **then**

    $E_{mul} = E_{l2} + 1$, $M_{align} = b'01 || M_{mul}$

else if $(M_{mul}(2 \cdot nf) = b'1)$ **then**

    $E_{mul} = E_{l2}$, $M_{align} = M_{mul}$

else

    $E_{mul} = E_{l2} - Position$, $M_{align} = b'01 || Shift(M_{mul}, Position)$

end if

{ *Step 5* : Comparing, final sign evaluation, exponent subtraction and mantissa swap}

$[g2_{A>B}, g2_{A=B}] = Compare(E_{mul}, C_e, M_{align}, C_m)$

$Sign_{sop} = Sign_{mul} \oplus C_s$

if $(g2_{A>B} = b'1) or (g2_{A=B} = b'1)$ **then**

    $M_{add_{l1}} = M_{align}$, $M_{add_{l2}} = C_m$

    $E_{sop} = E_{mul}$, $E_{add_{l2}} = E_{mul} - C_e$

else

    $M_{add_{l1}} = C_m$, $M_{add_{l2}} = M_{align}$

    $E_{sop} = C_e$, $E_{add_{l2}} = C_e - E_{mul}$

end if

{ *Step 6* : Shift and final mantissa determination}

$M_{shift} = b'0 || Shift(M_{add_{l2}}, E_{add_{l2}}, Right)$

if (Sign$_{sop}$=b'0) **then**

    $M_{sop} = M_{add_{l1}} + M_{shift}$

else

    $M_{sop} = M_{add_{l1}} - M_{shift}$

end if

{ *Step 7* : Leading-one-detection and normalization}

$Position = LOD(M_{sop})$

$X = Norm(Sign_{sop}, E_{sop}, M_{sop}, Position, n, ne, nf)$

In Algorithms 5-8, the right/left shifting and *LOD* functions as well as signed integer addition/subtraction and multiplication are frequently used. Exploration on these functions and operations helps to improve the performance of the floating-point units. Since the integer multiplication has greater significant impact on the performance of the floating-point units than the integer adder/subtractor, the integer multiplication, the design and analysis of the right/left shifting and *LOD* functions and the integer multiplication operations will be discussed in section 2.3.

2.3 Design and Enhancement of the Function and Operation

2.3.1 Leading-One-Detection based on Binary-Tree Algorithm

Leading-one-detection is a function applied to detect the first bit one from the most significant bit (*MSB*) to the last significant bit (*LSB*). Normally, *LOD* is performed by the comparison of two adjacent bits by a *Loop-Statement*. By means of this method, the critical delay is proportional to the length of the considered bit string, either 25-bits for the signal-precision IEEE standard representation or 52-bits for the double-precision IEEE standard representation. In order to reduce this critical delay, the binary-tree searching method has been employed instead of a conventional searching method such as *For-Statement*. The method [138] operates based on a hierarchy structure corresponding to an intermediate input binary string. The depth of this structure is determined by $log_2(N)$, where N is the size of an input binary string. The binary-tree structure is illustrated in Fig. 2.3.

Tab. 2.4 represents the truth table of the *Binary-Tree Cell* (*BT-Cell*), where X, C, $N_{v-pq} \in \{0, 1\}$ and p, q, $N_{loc-pq} \in \left\{0, ..\frac{N}{2} - 1\right\}$. Assuming that p and q are a coordinate in xy plane, where p is a position on the x axis and q is a position on the y axis (depth). Thus, *BT-Cell01* is a binary-tree cell located in depth level 0 and in column 1. $X_{N-1:0}$ is an input binary vector and C determines the direction of the search (either *MSB-to-LSB* or *LSB-to-MSB*). N_{loc-pq} is a selected location corresponding to the coordinates p and q. N_{v-pq} is the bit value of the selected location. For instance, in the 6^{th} case, X_N, X_{N-1} and C are equal to b'101, meaning that the *BT-Cell* searches the first bit one from *MSB* to *LSB*. Afterwards, N_{v-pq} is set to b'1 and N_{loc-pq} is equal to N.

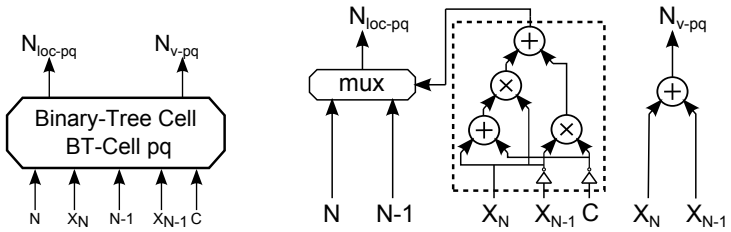
The corresponding optimised combinational logic is expressed by Equations (2.1) and (2.2). Fig. 2.2 depicts the design and architecture of the *BT-Cell*, where $\otimes$ is AND gate and $\oplus$ is OR gate.

$$N_{loc-pq} = \begin{cases} N, & \text{if } X_N \cdot (\overline{X_{N-1}} + \overline{C}) + \\ & (\overline{X_{N-1}} \cdot \overline{C}) \text{ is true} \\ N-1, & \text{otherwise} \end{cases} \tag{2.1}$$

Tab. 2.4: The binary selection algorithm for the *BT-Cell* implementation.

Case	X_N	X_{N-1}	C	$Node-loc.$ (N_{loc-pq})	$Node-Value$ (N_{v-pq})
1	b′0	b′0	b′0	N	b′0
2	b′0	b′0	b′1	N-1	b′0
3	b′0	b′1	b′0	N-1	b′1
4	b′0	b′1	b′1	N-1	b′1
5	b′1	b′0	b′0	N	b′1
6	b′1	b′0	b′1	N	b′1
7	b′1	b′1	b′0	N	b′1
8	b′1	b′1	b′1	N-1	b′1

$$N_{v-pq} = X_N + X_{N-1} \tag{2.2}$$



(a) BT-Cell (b) Node-Index and Node-Value

Fig. 2.2: The *Binary-Tree Cell* and internal logical architecture

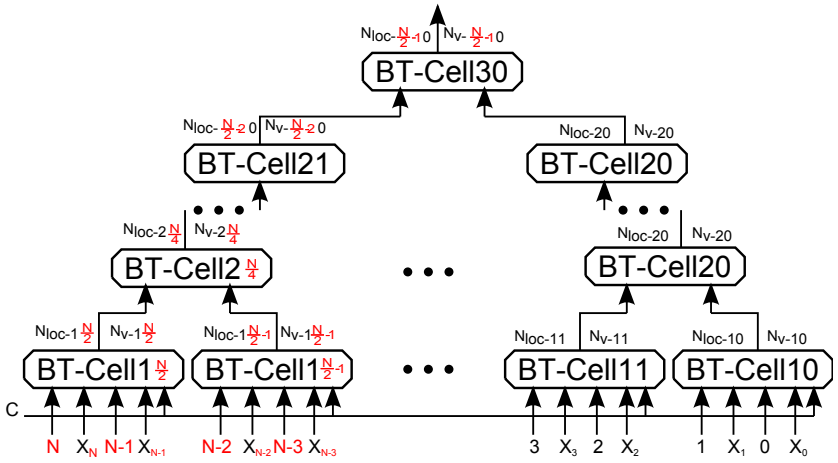


Fig. 2.3: The binary-tree structure

Fig. 2.4 illustrates the performance comparison of *LOD* performed by the *For-Loop method* and the *Binary-Tree method*, where the length of the input binary string (N) varies

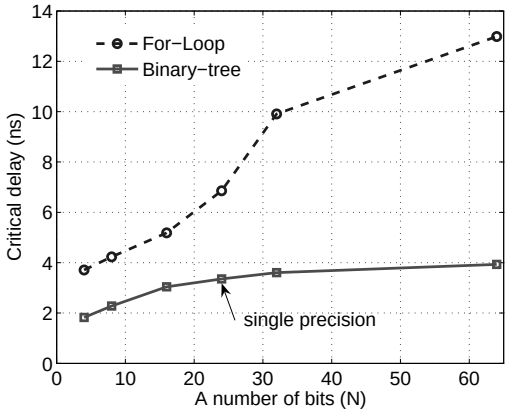


Fig. 2.4: Performance comparison between *For-Loop method* and *Binary-Tree method* based on the Xilinx Vertex 5 xc5vlx110t-3ff-1136 FPGA technology.

from 5 to 64. The graph shows that when N increases , the critical delay of the *For-Loop method* dramatically increases. On the other hand, the critical delay of the *Binary-Tree method* increases only slightly. Therefore, *LOD* based on the binary-tree architecture improves the performance of the floating-point unit significantly.

2.3.2 Right/Left Shifting function

The function is normally performed by a sequential shift-register, where the shifting length and the shifting direction are configurable. Similar to the critical delay analysis of *LOD*, the critical delay of the sequential shift-register is proportional to the maximum shifting length of a mantissa faction. In order to alleviate this critical delay, a multiplexer-based shift-register is proposed. Assuming that n is the maximum shifting length of the registers A and B, m is the location of the *LOD* in the register A and B. *sel* is an intermediate shifting length, where the shifting length is greater or equal to *n-1*. The number of utilized multiplexers is equal to n. Thus, the right-shifting multiplexer (*RMux*) function and the left-shifting multiplexer (*LMux*) function can be described by Equations (2.3) and (2.4).

$$b(m) = RMux_x = \begin{cases} a(0) & sel = 0 \\ a(1) & sel = 1 \\ \vdots & \vdots \\ a(n-1) & sel = n-m-1 \\ 0 & sel > n-m-1 \end{cases} \tag{2.3}$$

$$b(m) = LMux_x = \begin{cases} a(m) & sel = 0 \\ a(m-1) & sel = 1 \\ \vdots & \vdots \\ a(n-m-1) & sel = m-1 \\ 0 & sel > m-1 \end{cases} \tag{2.4}$$

Fig. 2.5 shows the performance of the multiplexer-based shifting function,, where the shifting length varies from 5 to 64 bits. The critical delay of the sequential shift-register is between 4.5 ns and 6.2 ns whereas the multiplexer-based one maintains the critical delay between 4 ns and 4.5 ns. Thus, the multiplexer-based shift-register achieves higher performance than the sequential shift-register.

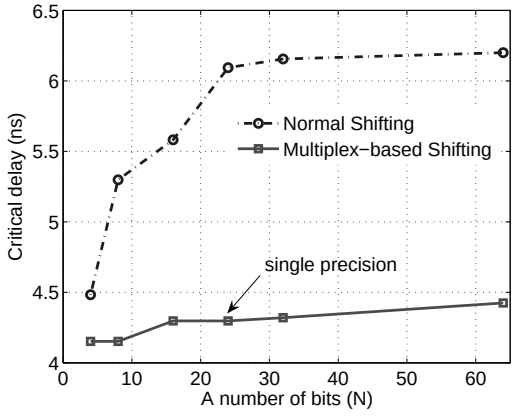


Fig. 2.5: The performance of the multiplexer-based shift function, where the shifting length is varied from 5 to 64 based on the Xilinx Vertex 5 xc5vlx110t-3ff-1136 FPGA technology

2.3.3 Partial Linear Integer Multiplier based on Pipelining Architecture

Since the main critical delay of the floating-point multiplier, the floating-point *PoS*, and the floating-point *SoP* come from an integer multiplier, shortening this delay is the objective of this section. The partial linear integer multiplier technique is applied to the multiplier's nominator. The pipeline architecture is utilized to improve the performance of the multiplier based on the amount of pipeline stages. n and m are denominated as the number of bits of the denominator and the number of partitions. The partial linear integer multiplier based on the pipeline architecture is illustrated in Fig. 2.6 with $m = 3$. The minimum number of pipeline stages is equal to $m + 1$. Carry-ripple-adders (*CRAs*) are employed to add the results from each partial multiplier generated by the previous stage. As reported in Tab. 2.5, the pipeline partial linear integer multiplier is synthesized in order to illustrate the trade-off between the consumed resources and the critical delay.

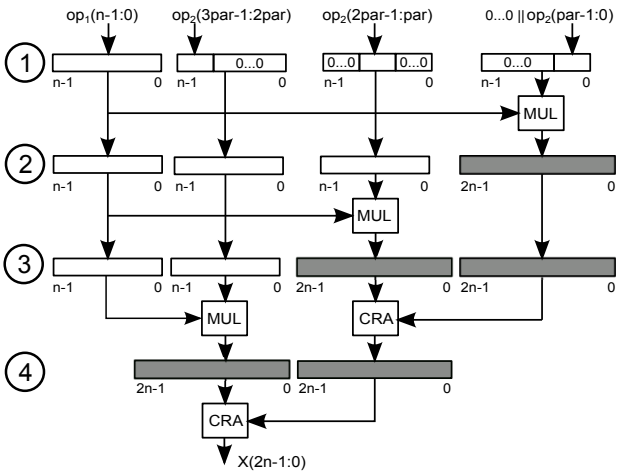


Fig. 2.6: The partial linear integer multiplier, where *m=3*

Tab. 2.5: Synthesis result of the partial linear integer multiplier based on the pipelining architecture on the Xilinx Vertex 5 xc5vlx110t-3ff-1136 FPGA technology.

Resources	m=1	m=2	m=3	m=4
Slice Reg.	0	146	316	466
Slice LUTs	886	894	893	905
Critical Delay(ns)	9.781	5.529	4.503	4.412
Max. Frequency(MHz)	102.23	180.86	222.08	226.63

2.4 Implementation and Investigation of Floating-Point Operator

In this section, the implementation and accuracy analysis of the floating-point operation algorithms proposed in Algorithms 5-8 are illustrated. Based on a pipeline architecture, the synthesis results on the Xilinx Virtex FPGA technologies and the 130-nm silicon technology are reported in Tabs. 2.18-2.21. The details of the partitioning stages used for consideration on each operators are explained below:

2.4.1 Synthesis Result Corresponding to Stage Numbers

2.4.1.1 Floating-Point Adder

In consideration of Algorithm 5 having 5 steps, performance and efficiency from the architectural point of view by merging or separating the steps is analyzed, where the *FP-Adder* is investigated in the 4-, 5-, and 6-stages respectively. In 4-stage, the steps 1 and 2 of Algorithm 5 are merged together. In 6-stage, *LOD* and normalization on step 5 are separated.

2.4.1.2 Floating-Point Multiplier

In Algorithm 6, there are 3 cases for evaluation which are the 4-, 5-, and 6-stages. For all cases, the steps 1-3 have been grouped because their total complexity is lower than the integer multiplier's. Thus, the 3-stage *FP-Multiplier* become an initial model. To improve its performance, the integer multiplier is split as 2-and 3-stage.

2.4.1.3 Floating-Point PoS

In Algorithm 7 there are 7 steps to floating-point sum-of-product. Since the 1^{st} to 3^{rd} steps are grouped together, the 5-stage *FP-SoP* becomes an initial model for consideration. As an integer multiplier generates the longest critical path, the multiplier is partitioned by two and three. Thus, there are 3 cases for evaluation, the 5-, 6-, and 7-stages.

2.4.1.4 Floating-Point SoP

Like *FP-PoS*, the 1^{st} to 3^{rd} steps are grouped together , Algorithm 8 and an integer multiplier is partitioned by two and three. There are also 3 cases for evaluation, the 5-, 6-, and 7-stages.

2.4.2 Comparison and Statistical Analysis in Accuracy

In this section the proposed 5-stage *FP-Adder* method (Algorithm 5) is implemented by using *VHDL* and then synthesized based on the Xilinx Virtex IIP xc2vp30-7ff896 FPGA technology. The synthesis result is compared with 5-stage *FP-Adder* corresponding to the methods in [69] and [71]. As illustrated in Tab. 2.6, the proposed *FP-Adder* provides better area and time efficiency than the two existing *FP-Adder* methods.

Tab. 2.6: Area and time efficiencies of a 5-stage *FP-Adder*.

Module	Clock speed (MHz)	Area (Slices)
Xilinx IP [69]	120	510
FP-Adder [71]	127	394
Proposed FP-Adder	140	326

In addition, the area and time efficiency of the proposed *LOD* method are compared with the *LOD* methods in [71]. The comparison result is shown in Tab. 2.7, where the proposed *LOD* based on binary-tree method presents better efficiency than the current *LOD* method. To compare with the 3-stage *FP-Adder* method proposed by [11], the proposed

FP-Adder method can also provide the 3-stage *FP-Adder* by merging steps 1 to 3 of Algorithm 5. However, since the *FP-Adder* method in [11] is designed based on leading-zero-anticipator (LZA) and implemented on the 0.5 μm CMOS technology which is relatively old, it is not convenient for comparison with the proposed *FP-Adder* method. Tab. 2.8 illustrates the time efficiency of *LOD*. *Abed LOD* [1] introduced a leading-one-detection to evaluate the first bit one in 16-, 32-, and 64-bit by decoding technique, where a 4-bit *LOD* cell becomes a basic component for implementation of a higher-bit *LOD* module. The Abed LOD is implemented on the 0.6 μm CMOS technology, however the proposed *LOD* is introduced on FPGA technology. Notice that the performance of the proposed *LOD* considered on FPGA-based technology is nearly the same as the performance of the *Abed LOD* on the CMOS 0.6 μm technology. Tabs. 2.9, 2.10, and 2.11 list the published floating-point operators implemented on FPGA-based and CMOS-based technologies.

Tab. 2.7: Area and time efficiency of *LOD*.

Module	Critical Delay (ns)	Area (Slices)
LOD [71]	8.32	14
Proposed LOD	5.726	147

Tab. 2.8: Time efficiency of the published and proposed *LOD* methods.

Method	Bit-width	Technology	Max. Frequency
Abed LOD [1]	16	CMOS 0.6 μm	310 MHz
	32		265 MHz
	64		215 MHz
Proposed LOD	16	FPGA technology	333 MHz
	32	Vertex 5	300 MHz
	64	XC5VLX100t-3FF1136	250 MHz

Tab. 2.12: Statistical error comparisons of float-point operators simulated based on their hardware and Matlab/Simulink models with input operands varied from $-10^{38.532}$ to $10^{38.532}$.

Operator	Max. ε	Min. ε	$\lvert\bar{\varepsilon}\rvert$	$\sigma(\varepsilon)$
FP-Adder	1.0571E-6	3.7213E-12	1.6678E-7	1.6528E-7
FP-Mult.	1.4687E-6	3.8625E-15	1.5056E-7	1.9464E-7
FP-PoS	1.3892E-3	7.9421E-13	1.7340E-4	2.0198E-4
FP-SoP	3.5168E-4	6.8965E-10	4.6439E-5	4.4634E-5

For the computation precision analysis, the floating-point arithmetic units are implemented in VHDL conforming to the proposed Algorithms 5-8. The results are compared

Tab. 2.9: Floating-point adder information of the published articles based on FPGA and CMOS technologies.

Articles	Method	Technology	bit-width	stage	Delay (ns)	Area/resource
Tsen et al. [117]	Binary integer decimal-based	CMOS 0.11 μm	64	≥ 2	2.4	0.55 μm^2
Narasimban et al. [85]	general-purpose	Xilinx FPGA XC4005	64	8	16.0	82% of resource utilization
Malik et al. [72]	far and close data-path	Xilinx FPGA Virtex 2	32	2	27.059	541 Slices
Karlström et al. [57]	general-purpose	Xilinx FPGA Virtex 4	32	8	27.059	846 Slice LUT 429 Slice FF
CoreGen [131]	general-purpose	Xilinx FPGA Virtex 5	32	12	2.5	429 Slice LUT 561 Slice FF
Grushin [39]	improving normalization	CMOS 0.13 μm	32	4	1.25	6,127 μm^2
Proposed	Improved LOD and Shifter	CMOS 0.13 μm	32	4	0.82	16,747 μm^2
				5	0.8	18,226 μm^2
				6	0.18	19,396 μm^2
		Xilinx Virtex 5	32	4	4.375	200 Slice Reg. 442 Slice LUT. 187 Slice FF.
				5	3.605	241 Slice Reg. 483 Slice LUT. 200 Slice FF.
				6	3.168	290 Slice Reg. 488 Slice LUT. 255 Slice FF.

Tab. 2.10: Floating-point multiplier information of the published articles based on FPGA and CMOS technologies.

Articles	Method	Technology	bit-width	stage	Delay (ns)	Area/resource
Raafat et al. [98]	IEEE 754-2008 Decimal Parallel	CMOS 0.18 μm	32	4	2.62	994,081.4 μm^2
				5	2.2	1,160,125 μm^2
				6	1.94	1,187,318.6 μm^2
Yu [134]	Booth integer multiplier	CMOS 0.5 μm	64	3	6.0	no information
Baesler et al. [10]	IEEE 754-2008 Decimal Parallel	Xilinx Virtex 5 built by Xilinx COGEN	64	5	25.2	2,284 Slice Reg. 2,297 Slice LUT. 1,989 Slice FF.
Makino et al. [70]	general-purpose	CMOS 0.5 μm	64	4	3.5	21.42 mm^2
Yamada et al. [132]	LZA with Booth integer multiplier	CMOS 0.3 μm	64	2	13.3	no information
Aty et al. [8]	general-purpose	FPGA Virtex II	32	3	25.06	1,088 Slice 186 Flip-Flop
Shiann-Rong et al. [109]	Low-power by ignoring rounding	CMOS 0.18 μm TSMC	32	2	5.2	153,039 μm^2
			64	2	8.0	574,268 μm^2
CoreGen [131]	general-purpose	Xilinx Virtex 5	32	9	2.22	88 Slice LUT 117 Slice FF 3 DSP48E
Karlström [57]	general-purpose	Xilinx Virtex 4	32	8	2.7	189 Slice LUT 154 Slice FF 4 DSP48E
Proposed	Improved LOD, Shifter and integer multiplier	CMOS 0.13 μm	32	4	0.95	39,487 μm^2
				5	0.9	40,413 μm^2
				6	0.83	47,900 μm^2
		Xilinx Virtex 5	32	4	5.620	332 Slice Reg. 1,132 Slice LUT. 209 Slice FF.
				5	4.503	395 Slice Reg. 1,159 Slice LUT. 253 Slice FF.
				6	4.412	599 Slice Reg. 1,155 Slice LUT. 282 Slice FF.

Tab. 2.11: Floating-point *SoP* information of the published articles based on FPGA and CMOS technologies.

Articles	Method	Technology	bit-width	stage	Delay (ns)	Area/resource
Dinechin et al. [27]	general-purpose	Xilinx Virtex 4	32	13	2.75	319 Slices 4 DSP48E
CoreGen [131]	general-purpose	Xilinx FPGA Virtex 4	32	26	2.73	484 Slices 8 DSP48E
Vangal et al. [121]	Integer multiplier with Wallace tree	CMOS 0.09 μm	32	12	0.2	2.0724 mm^2
Paidimarri [90]	Booth encoding integer multiplier	Altera FPGA Stratix3	32	11	3.8	3437 Area LUTs
Jain et al. [54]	general-purpose	CMOS 0.065 μm	32	8	0.29	0.54 mm^2
Proposed	Improved LOD and Shifter	CMOS 0.13 μm	32	5	1.41	49,114 μm^2
				6	1.2	53,470 μm^2
				7	1.05	63,799 μm^2
		Xilinx Virtex 5	32	5	5.962	450 Slice Reg. 1,574 Slice LUT. 297 Slice FF.
				6	5.662	492 Slice Reg. 1,669 Slice LUT. 325 Slice FF.
				7	4.937	783 Slice Reg. 1,634 Slice LUT. 422 Slice FF.

with those of the hardware VHDL simulation in 32-bit single-precision computation and of Matlab/Simulink in double-precision computation. The computation errors are considered and reported in statistical terms. The maximum error (Max. ε), the minimum error (Min. ε), the absolute error ($|\bar{\varepsilon}|$), and the standard deviation ($\sigma(\varepsilon)$) are listed in Tab. 2.12. For the testing environment, the value of three input operands 1 varied from $-10^{38.532}$ to $10^{38.532}$. From the statistical error comparison, the computational result performed by single-precision floating-point arithmetic units in hardware provides the computational accuracy close to the computational results calculated by the Matlab/Simulink in 32-bit single-precision computation. The computational error may be derived from rounding computation which is not considered in the proposed floating-point algorithms.

2.5 Design and Architecture of Floating-Point Arithmetic Accelerator

The floating-point operators, i.e. *FP-Adder*, *FP-Multiplier*, *FP-PoS*, and *FP-SoP*, which have been designed and analysed in the previous sections are combined into a floating-point accelerator. The architecture of the accelerator is illustrated in Fig. 2.7. The integration of the accelerator into a multi-processor system is depicted in Fig 2.8.

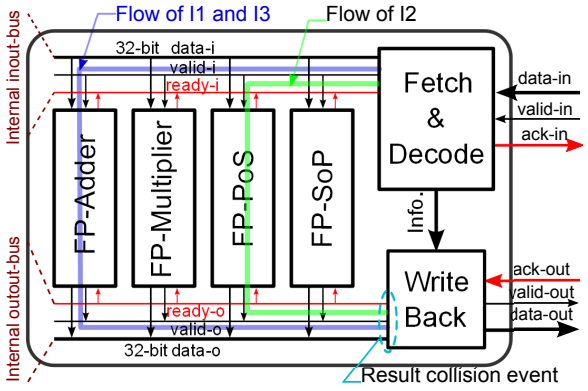


Fig. 2.7: The architecture of the floating-point accelerator consisting of *FP-Adder*, *FP-Multiplier*, *FP-PoS*, and *FP-SoP*.

2.5.1 Design and Architecture

In Fig. 2.7, the 32-bit pipelined instruction architecture is employed for the design of the floating-point accelerator in order to fetch and execute an intermediate instruction in every clock cycle. The design consists of three stages : *Fetch-and-Decode*, *Execute* and *Write-back*. The signals *data-in/out*, *valid-in/out* and *ack-in/out* are defined as external inputs and outputs connected to *Bus A* and *Bus B*. The 2-bit *ack-in/out* signal is used to notify the received status to a source module, where b'00 shows the status that the destination is

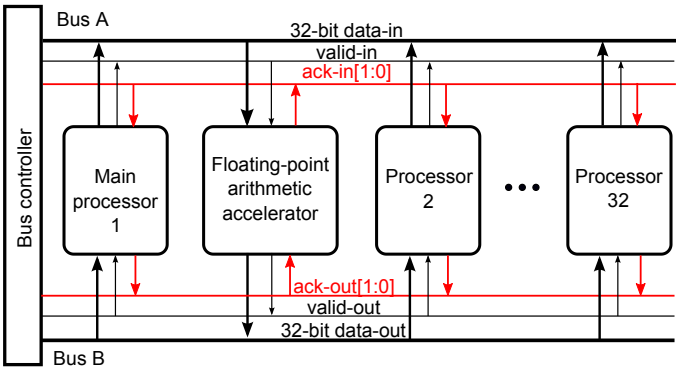


Fig. 2.8: The architecture of the floating-point accelerator cooperating with multiple processors.

Tab. 2.13: The micro-instruction of the proposed floating-point accelerator for any general purpose processor.

Cmd	Mnemonic	Operand	Operation	Description	#Clock
x'0001	*FPADD32*	I_d, P_{id}, op_1, op_2, R	$R \leftarrow op_1 + op_2$	Addition	10
x'0002	*FPMUL32*	I_d, P_{id}, op_1, op_2, R	R$\leftarrow op_1 \times op_2$	Multiplication	10
x'0003	*FPPoS32*	I_d, P_{id}, op_1, op_2, op_3, R	R$\leftarrow (op_1+op_2) \times op_3$	Product-of-Sum	13
x'0004	*FPSoP32*	I_d, P_{id}, op_1, op_2, op_3, R	R$\leftarrow (op_1 \times op_2) + op_3$	Sum-of-Product	13

in ready stage; b'01, b'10, and b'11, inform that the 1^{st}, 2^{nd}, and 3^{rd} words respectively are accepted by the destination. The internal *ready-i/o* signal indicates that the destination is in the ready stage. Like the hand-checking protocol, Figs. 2.11 and 2.12 show the timing diagram of the *Fetch-and-Decode* and *Writeback* cycles. Fig. 2.8 illustrates the diagram, where the proposed floating-point accelerator is applied to a multi-processor system. The processors can send and receive data to and from the accelerator via a bus-system, *Bus A* and *Bus B*. A bus controller is employed to handle any requests to the two buses at runtime.

The proposed floating-point accelerator is targeted to operate at the maximum frequency on the Xilinx Virtex 5 device xc5vlx110t-3ff-1136 FPGA technology (200 MHz) and at 1 GHz using the 130-nm silicon technology. Consequently, the corresponding number of stages utilized in the design of *FP-Adder*, *FP-Multiplier*, *FP-PoS*, and *FP-SoP* are 5-stages, 5-stages, 7-stages and 7-stages, respectively in accordance with the synthesis results given in Tabs. 2.18-2.21.

2.5.2 Micro-Instruction and Timing Diagram

The micro-instruction pattern and the timing diagram of the floating-point accelerator are presented in order to simplify to any general propose processors. From the proposed floating-point operators, three types of instruction and reply format, i.e. short and long (*#F1* and *#F2*) and write-back (*#R1*), are introduced in Fig 2.9. The short and long in-

struction formats are respectively 3 and 4 words long. The 1^{st} word consists of a 16-bit command (*cmd*), 8-bit instruction ID (I_{id}) and 5-bit processor ID (P_{id}). Up to 32 processors can be thus supported. The 2^{nd} to 4^{th} words are the three operands of the floating-point operation. There are 2 words for the reply format *#R1*, where the 1^{st} word comprises an 8-bit instruction ID (I_{id}) and 5-bit processor ID (P_{id}). The last word is an intermediate result performed by the floating-point units. Tab. 4.9 represents the four micro-instruction tables to be employed by any general purpose processors.

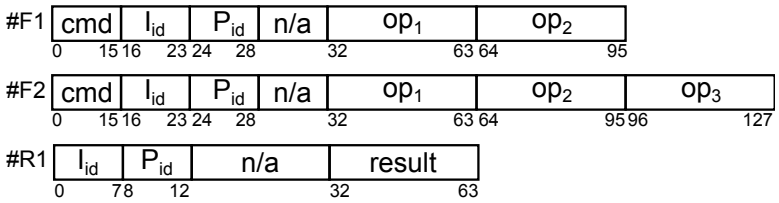


Fig. 2.9: Instruction format *#F1*, *#F2* and Reply format *#R1* of the accelerator

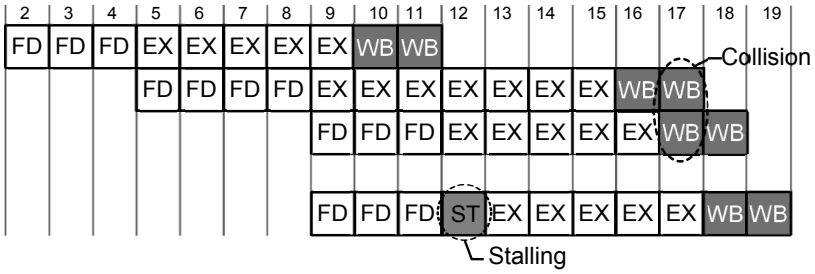


Fig. 2.10: Result collision when either *FPPoS32* or *FPSoP32* instruction are first required and followed by either *FPADD32* or *FPMUL32*, *FD*, *EX*, and *WB* are *Fetch-and-Decode* cycle, *Execution* cycle, and *WriteBack* cycle.

Fig. 2.10 shows the result of a collision event at the internal output-bus when a long instruction is ahead of a short instruction. *FPADD32*, *FPPoS32*, and *FPMUL32* are called in sequence starting from the 2^{nd} clock cycle. The resulting collision event happens in the 17^{th} clock cycle, where two word results performed by *FPPoS32* and *FPMUL32* have been written back simultaneously. In order to alleviate this problem, a simple pattern instruction format detection is introduced into the *Fetch-and-Decode* module. If the previous instruction is a long instruction and the current instruction is a short instruction, a stalling stage (*ST*) is inserted. The resulting timing diagram using the ST stage is illustrated in Fig. 2.10, where the 1^{st} and 2^{nd} *WB* generated from the *FPMUL32* are presented at the 18^{th} and 19^{th} clock cycles.

The timing diagram in Fig. 2.11 shows the execution of three instructions, *I1*, *I2*, and *I3* which are *FPADD32*, *FPPoS32*, and *FPMUL32* respectively. Each word presented by the *data-in* signal will be validated by the *valid-in* signal. Whenever destination has already received the computation result, the 2-bit *ack-in* signal will be incremented. It is then reset when the result is completely received.

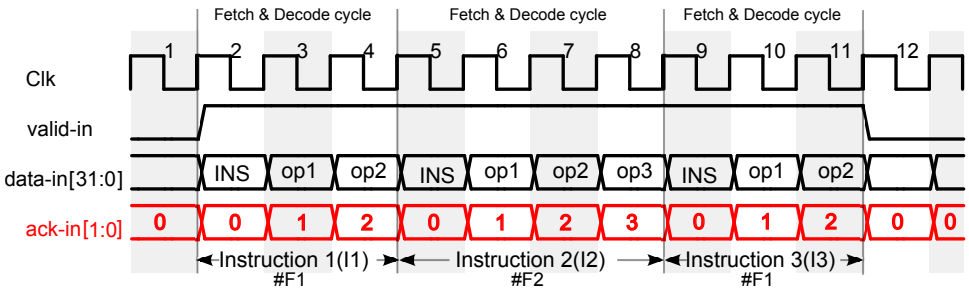


Fig. 2.11: The timing diagram illustrates an event of three input instructions, *I1*, *I2*, and *I3*, representing the internal input-bus in *Fetch-and-Decode* cycle.

The timing diagram in Fig. 2.12 shows the *Writeback* cycle of the personal information (Info.) and the computation result generated by *I1*, *I2*, and *I3*. With always active the *ready-o* signal, the computation result and its validation are presented on the *data-o* signal and on the *valid-o* signal at 10^{th}, 16^{th}, and 18^{th} clock cycles. Considering at 10^{th} clock cycle, as soon as the *valid-o* signal is active, the personal information of *I1* is written to the *data-out* signal, connected to the 32-bit *data-out* signal on Bus B. It is followed by the computation result on the next clock cycle, where the *valid-o* signal is also active for two clock cycles. In common with the *Writeback* cycle of *I1*, the personal *Info.* and the computation results of *I2* and *I3* are presented at the 16^{th} and 18^{th} clock cycles.

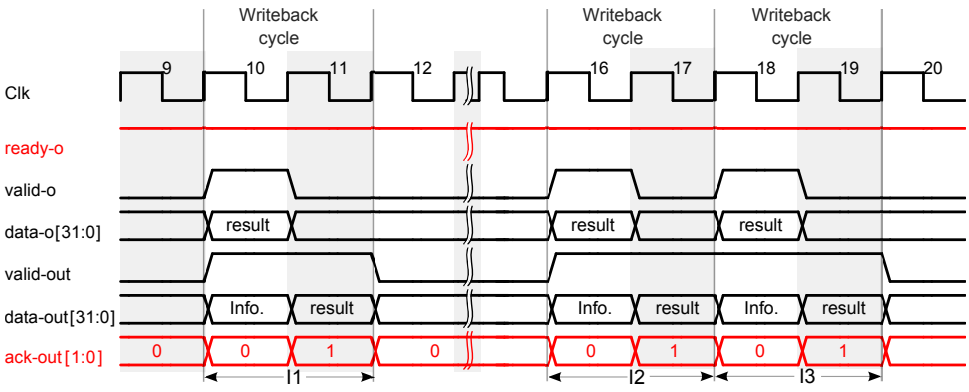


Fig. 2.12: The timing diagram of the personal information and the computational results of *I1*, *I2* and *I3* on their *Writeback* cycle.

2.5.3 Performance Analysis

The performance of the proposed floating-point accelerator is evaluated by *Fetch Instruction Rate* (instr./s) and throughput in the number of floating-point operations per sec (*FLOPS*). f is presented as the maximum operating frequency of the design. The *Fetch Instruction Rate* (*FR*) means the number of floating-point instructions able to be fetched and decoded in a certain period. Obviously, the *FR* depends on accelerator's architecture, where if system data-width equals the defined data-word, the *FR* will equal f. However, in our design the accelerator has 32 bits data-width, where 1 data-word is 32 bits.

There are also two types of input instruction formats #*F1* and #*F2* which provide 3 and 4 data-words for one floating-point operation. Thus, the maximum and minimum *FR* are determined by $f/3$ and $f/4$.

Tab. 2.16 shows the *Fetch Instruction Rate* and the throughput of the proposed design based on the Xilinx xc5vlx110t-3ff-1136 FPGA and 130-nm silicon technologies, and also exhibits an example for the floating-point unit selection when the target frequency response is at 200 MHz for FPGA-based and at 1 GHz for silicon-based. To achieve the goal, the 5-stage *FP-Adder* and *FP-Multiplier* as well as the 7-stage *FP-PoS* and *FP-SoP* from Tabs. 2.18 and 2.21 are selected.

Tab. 2.14: Synthesis result on the FPGA Virtex 5 xc5vlx110t-3ff-1136 technology.

	Utilization	% of Total
Number of slice registers	1973 out of 69,120	3%
Number of slice LUTs	4946 out of 69,120	7%
Number of slice LUT-FF	92 out of 385	24%
Number of BUFG/BUFGCTRLs	9 out of 32	3%
Critical Delay	0.5 ns	
Maximum Frequency	200 MHz)	

Tab. 2.15: Synthesis result on the 130-nm silicon technology.

	Utilization
Area(um)	189513
Power(mW)	60.607
Critical Delay	1 ns
Maximum Frequency	1 GHz)

Tabs. 2.14 and 2.15 summarizes the consumed resources and areas of the floating-point accelerator when synthesized on the Xilinx xc5vlx110t-3ff-1136 FPGA technology and on the 130-nm silicon technology targeting at 200 MHz and at 1 GHz, respectively.

Tab. 2.16: Performance definition and evaluation on the Xilinx Virtex5 xc5vlx110t-3ff-1136 FPGA and 130-nm silicon technologies at 200 MHz and 1 GHz

Mesurment		FPGA	Silicon
Max. Fetch Rate (Minstr./s)	$f/3$	66.67	333.33
Min. Fetch Rate (Minstr./s)	$f/4$	50	250
Max. Writeback Rate (Mrpy./s)	$2f/3$	133.33	666.67
Min. Writeback Rate (Mrpy./s)	$f/2$	100	500
Max. Throughput (MFLOPS)	Max. WR$/2$	66.67	333.33
Min. Throughput (MFLOPS)	Min. WR$/2$	50	250

WR = Writeback Rate, Minstr = Mega instr., Mrpy = Mega reply

Tab. 2.18: Hardware synthesis results of *FP-Adder* and *FP-Multiplier* on the Xilinx Virtex5 xc5vlx110t-3ff-1136 FPGA technology.

Operator	*FP-Adder*			*FP-Multiplier*		
Resources	4-stage	5-stage	6-stage	4-stage	5-stage	6-stage
Slice Reg.	200	241	290	332	395	599
Slice LUTs	442	483	488	1,132	1,159	1,155
LUT-FF pairs	187	200	255	209	253	282
Critical Delay(ns)	4.375	3.605	3.168	5.620	4.503	4.412
Max. Freq.(MHz)	228.60	277.40	315.63	177.95	222.09	226.63

Tab. 2.19: Hardware synthesis results of *FP-PoS* and *FP-SoP* on the Xilinx Virtex5 xc5vlx110t-3ff-1136 FPGA technology.

Operator	*FP-Product-of-Sum*			*FP-Sum-of-Product*		
Resources	5-stage	6-stage	7-stage	5-stage	6-stage	7-stage
Slice Reg.	317	435	500	430	492	783
Slice LUTs	1642	1438	1463	1,574	1669	1634
LUT-FF pairs	260	306	337	297	325	422
Critical Delay(ns)	6.827	5.620	4.95	5.962	5.662	4.937
Max. Freq.(MHz)	146.48	177.95	202.23	167.73	177.95	202.54

Tab. 2.20: Hardware synthesis results of *FP-Adder* and *FP-Multiplier* on the 130-nm silicon technology.

Operator	*FP-Adder*			*FP-Multiplier*		
Resources	4-stage	5-stage	6-stage	4-stage	5-stage	6-stage
Area(um).	16,747	18,226	19,396	39,487	40,413	47,900
Power(mW)	8.5772	10.318	18.791	14.5492	16.377	21.783
Critical Delay(ns)	0.82	0.8	0.48	0.95	0.9	0.83
Max. Freq.(GHz)	1.22	1.25	2.083	1.05	1.11	1.20

Tab. 2.21: Hardware synthesis results of *FP-PoS* and *FP-SoP* on the 130-nm silicon technology.

Operator	*FP-Product-of-Sum*			*FP-Sum-of-Product*		
Resources	5-stage	6-stage	7-stage	5-stage	6-stage	7-stage
Area(um).	44,807	52,104	54,897	49,114	53,470	63,799
Power(mW)	16.99	24.923	25.716	14.692	15.510	26.193
Critical Delay(ns)	1.12	0.9	0.85	1.41	1.2	0.95
Max. Freq.(GHz)	0.89	1.11	1.18	0.71	0.83	1.05

2.6 Summary

This chapter proposes the design and analysis of floating-point operators and accelerator in compliance with the floating-point IEEE standard format. The operators are considered in the algorithmic form for hardware simplification. The standard and non-standard floating-point operators, i.e. *FP-Adder*, *FP-Multiplier*, *FP-PoS*, and *FP-SoP*, are analysed in order to increase the performance. The *FP-PoS* and *FP-SoP* algorithms are also introduced by the fusion of the floating-point addition/subtraction and multiplication algorithms. The leading-one-detection based on binary-tree architecture and the multiplexer-based right/left shifting architecture are proposed to alleviate the restriction derived from the maximum critical delay corresponding to the longest critical path. Additionally, the partial architecture of integer multiplier is introduced and analysed in order to improve the performance. The floating-point algorithms are implemented in *VHDL*, synthesized, and simulated and their computation accuracy are statistically compared with the ideal results from Matlab/Simulink. The results illustrate that the proposed operators provide high performance and high accuracy.

Moreover, the floating-point accelerator is designed by grouping the introduced floating-point operators. The maximum operating frequencies of 200 MHz on the Xilinx Virtex5 xc5vlx110t-3ff-1136 FPGA technology and at 1 GHz on the 130-nm silicon technology become the design constraint. To simplify for any general purpose processors, the micro-instruction set is introduced, where its maximum and minimum clock delay at 10 and 13 clock cycles for the short instruction format *#F1* and the long instruction format *#F2* are reported. From evaluation results, the accelerator provides the maximum and minimum instruction rate at 66.67 and at 50 Minstr./s on *FPGA* and at 333.33 and at 250 Minstr./s on Silicon. Its maximum and minimum throughput are at 66.67 and at 50 *MFLOPS* on *FPGA* and at 333.33 and at 250 *MFLOPS* on silicon.

Chapter 3

CORDIC Algorithm and Elementary Functions based on Non-Redundant Method

Contents

In this chapter, rotation-extension CORDIC methods, i.e. double-rotation and triple-rotation, are proposed for the objectives of improving the performance, time efficiency, and computational accuracy of the CORDIC algorithm in radix-2. In the double-rotation and triple-rotation methods, the convergences of the CORDIC are accelerated by duplicating and triplicating the micro-rotation angles to be 2θ and 3θ, respectively. The non-redundant mechanism, which a rotation direction δ is in a set of 1 and -1, depending on an intermediate converging parameter (either y or z), is applied to constant scaling factors. Convergence ranges and computational accuracies of elementary functions performed by the proposed CORDIC methods in rotation mode and vectoring mode on the circular, hyperbolic, and linear coordinate systems are examined, investigated and compared with Matlab/Simulink ideal results. The comparison results show that the proposed CORDIC methods provide higher accuracy than the conventional CORDIC at the same number of iterations. A high precision CORDIC algorithm is introduced for simple *VLSI* implementation. Moreover, extension functions derived from CORDICs' elementary functions in hyperbolic coordinate system, i.e. natural logarithm and square root, are considered and investigated. Finally, convergence range problem of the CORDIC is discussed.

3.1 Introduction

The *COordinate Rotation DIgital Computer* (CORDIC) is known as an iterative algorithm using only shift-add operations to perform several mathematic functions for scientific and engineering applications. The CORDIC was first described in 1959 by J.E. Volder [124] as an elegant way to evaluate trigonometric functions. It was originally developed to replace the analog navigation computer on the B-58 aircraft bomber due to its requirement for high performance and accuracy [125]. In 1971, J. Walther [126] extended the CORDIC algorithm to hyperbolic functions and the algorithm is presently used in many application areas such as matrix computation, digital signal processing, digital image processing, communication, robotics and graphics [75]. The key concept of the algorithm is based on

the rotation of a two-dimensional (2-D) (x,y) vector in circular, hyperbolic, and linear coordinate systems. J. E. Volder had described and implemented an iterative formulation based on the CORDIC for trigonometric, division and multiplication functions. CORDIC became more and more attractive when J. Walther illustrated that by modifying a few parameters, it can be used to perform a single algorithm for unified implementation of a wide range of elementary transcendental functions related to logarithms, exponentials, square root, etc. Meanwhile, D. Cochran [21] introduced various algorithms and examples which have proven that the CORDIC technique was a better choice for scientific calculating applications. Its popularity has been increased thereafter primarily due to the potential for an efficient and low-cost implementation. Some of the popular applications are: (1) direct digital frequency synthesis [33], digital modulation and coding for speech/music synthesis and communication; (2) direct and inverse kinematics computation for robot manipulation; and (3) planar and three-dimensional (3-D) vector rotation for graphics and animation [32].

The development of the CORDIC algorithm and architectures [63] is taken place to achieve the highest throughput rate and to reduce the hardware-complexity as well as the computational latency on *VLSI* implementation. Some of the typical approaches for reducing-complexity implementation are targeted on minimization of the scaling-operation and complexity of barrel-shifters and adders in the CORDIC engine. The inherent drawbacks of the conventional CORDIC algorithm are latency and accuracy of computation. Angle recording schemes, mixed-grain rotation and higher radix CORDIC have been developed to reduce the latency. Parallel and pipelined techniques have been suggested for high-throughput computation. In this chapter, five subjects are discussed:

1) Design and analysis of the CORDIC methods which can improve the computational accuracy and latency.

2) Introduce a verification method which can be applied for verification and investigation of the CORDIC methods in practice.

3) Introduce a simple unified micro-rotation for *VLSI* implementation.

4) Consider the possibilities, where the CORDIC methods can be applied for other elementary functions such as natural logarithm and square root.

5) Study the problems of the CORDIC method when dealing with the convergence range and the proposed solution.

3.2 State-of-Art

There are various proposed methods to improve performance, to reduce computational latency and to increase computational accuracy of the CORDIC algorithm, i.e.

1) High radix method,

2) Parallel rotation method,

3) Redundant number representation,

4) Rotation extension method.

3.2.1 High Radix CORDIC method

In 1996, E. Antelo et al. [3] proposed the mixed radix-2 and radix-4 methods for a redundant CORDIC processor implemented in pipeline architecture. The combination of the two radixes can reduce the computational latency and area of the pipeline stage. The full radix-4 and very-high CORDIC methods with on-line scaling factor computation for high performance rotation architectures [5] [4] were introduced. Meanwhile, C. C. Li [68] suggested a fast on-line scaling factor compensation to the redundant CORDIC algorithm in radix-4, and simplified the on-line decomposition algorithm in hardware. J. Villalba [123] introduced an extension version of the radix-4 CORDIC method in vectoring mode, where the selection method is applied for non-redundant and redundant computations. The constant scaling factor of the radix 2-4-8 CORDIC algorithm was investigated by T. Aoki et al. [6]. The algorithm dynamically changed from radix-2 to radix-4 and from radix-4 to radix-8 depending on the specified sequences of the running CORDIC algorithm. The constant scaling factor of each radix was expressed as $K_c = \prod_{i=1}^{N} \sqrt{1 + \delta_i^2 \cdot r^{-2 \cdot i}}$, where K_c is a constant scaling factor, N is the maximum number of iterations, δ_i is a rotation direction of micro-rotation of CORDIC, and r_i is the radix-based determined by i^{th} iteration. The articles dealing with the high radix CORDIC method show that the scaling factor applied for computation and compensation becomes a main problem. The scaling factor cannot be fixed as a constant value due to the redundant value corresponding to the high radix, leading to be unstable in computation. They attempt to solve this problem by proposing the on-line scaling factor computation method and by finding a range of the computation in order to sustain the scaling factor on the radix-2-4-8 CORDIC algorithms. However, the CORDIC method used for the constant scaling factor is too complex for the hardware implementation.

3.2.2 Parallel CORDIC rotation method

The parallel CORDIC method was proposed by T. B. Juang et al. [118]. The algorithm predicts a rotation direction δ_i from a binary value of an initial angle. The original sequential CORDIC rotations was divided into two phases, where the rotation in each phase was executed in parallel. S. F. Hsiao [108] applied the method to implement a sine/cosine generator with memory-efficient concept; meanwhile W. Han et al. [43] implemented a digital down converter based on the parallel CORDIC algorithm. Thereby, the parallel

method can provide a constant scaling factor because the two parallel processing cores are based on the conventional CORDIC. Although, the scaling factor has been solved by using the parallel technique, the overheads of summation and prediction of the two parallel CORDIC engines and rotation direction are aggravated.

3.2.3 Redundant Number Representation Method

The third efficient way to accelerate the CORDIC algorithm is to use redundant number representation, such as signed-digit (*SD*) representation and carry-save representation. In 1990, M. D. Ercegovac [30] presented redundant (carry-free) addition instead of the conventional (carry-propagate) computation and the on-line scaling factor computation method in order to minimize the area and to increase the bandwidth. However, the on-line computation method is too complex which is very difficult to implement. By ignoring the problem dealing with the on-line scaling factor computation complexity, N. Takagi et al. [114] introduced the redundant CORDIC method with a constant scaling factor. Therefore, the scaling factor is selected to minimize the error from the CORDIC computation, where the computational accuracy is taken into account.

3.2.4 Rotation Extension Method

From the reviewed articles, the reduction of computational latency by the high radix method causes the scaling factor in an unstable situation due to the redundant value which corresponds to the selected radix. The unstable scaling factor situation can be solved by the on-line computation method which is too complex for hardware implementation. The parallel technique is another method applied to accelerate the CORDIC computation. This method is based on two conventional CORDIC cores processing in parallel. Thereby, the scaling factor problem is solved, but the prediction overheads of the rotation direction and the combination of the two conventional CORDIC results are increased. The extension-rotation methods based on radix-2 accelerate the micro-rotation angle ϕ with $n \cdot \phi$, where n is an integer value. The double-rotation method performs the computational results with micro-rotation angle 2ϕ. Its constant scaling factor can be performed either by on-line computation method or by selecting a constant value from error optimization on computational results.

The design and architecture of the CORDIC algorithm on the radix-2 are straightforward for hardware implementation. The rotation-extension CORDIC methods, i.e. non-redundant double-rotation and non-redundant triple-rotation, are proposed. Afterwards, they are analysed, investigated, and simulated in order to evaluate the constant scaling factors and the input domain capability for elementary functions. The initial parameter values x_{in}, y_{in}, and z_{in} and the compensation parameter values (K_s and K_s^{-1}) for each elementary function in circular, hyperbolic, and linear coordinate systems are also considered. The computational accuracies of the elementary functions are compared

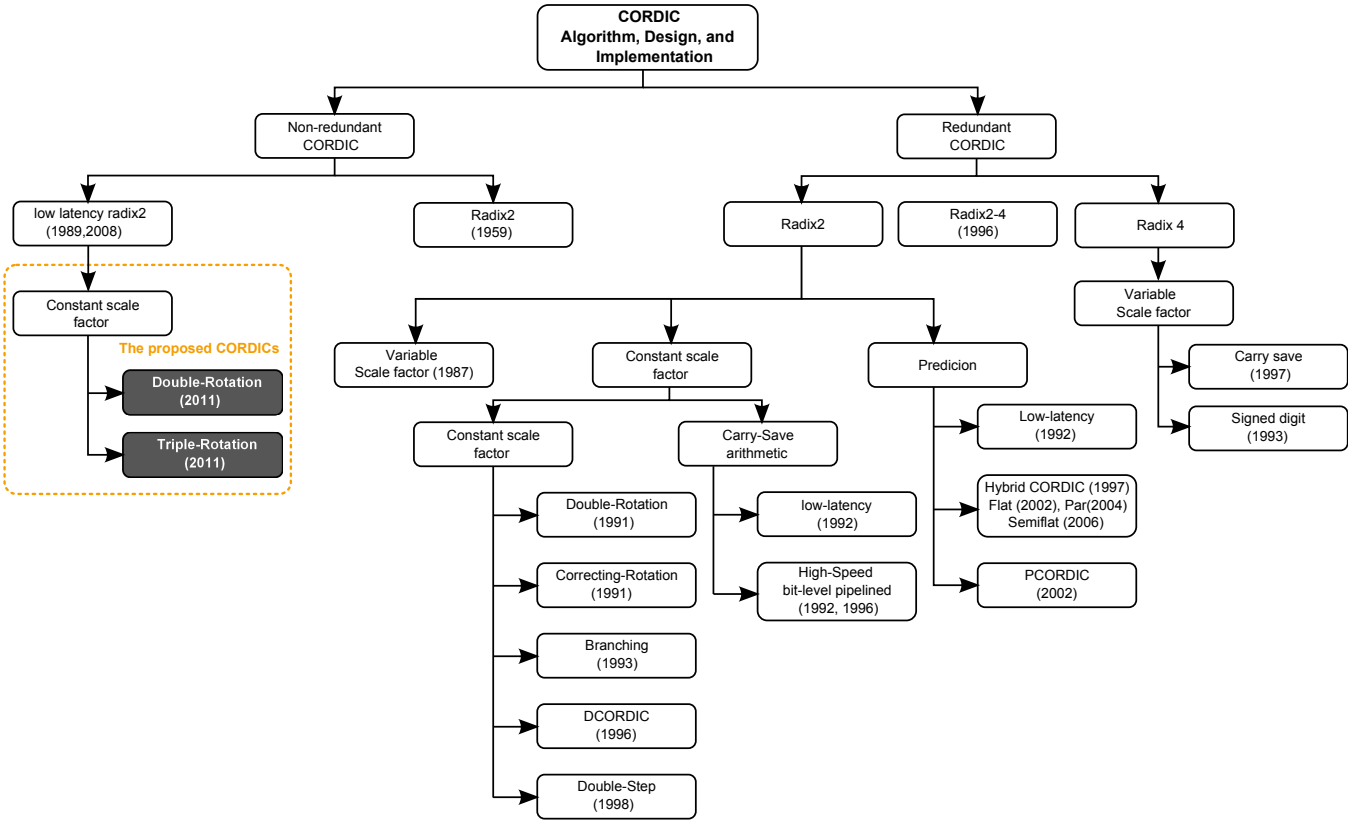


Fig. 3.1: Taxonomy of the CORDIC methods [63]

with Matlab/Simulink ideal results in order to show that the proposed CORDIC methods provide high computational accuracy. Moreover, extension functions derived from the CORDIC fundamental functions in hyperbolic coordinate system, i.e. natural logarithm and square root, are considered and investigated. Finally, the convergence range problem on the CORDIC's elementary functions is discussed and proposed for future research. The taxonomy of the CORDIC methods is depicted in Fig. 3.1, where the proposed methods are in a group of the low latency radix-2 with constant scaling factor.

3.3 Rotation-Extension CORDIC Algorithm

The algorithm normally bases on the rotation of a vector on the xy plane. As shown in Fig. 3.2, a vector $x_{in}y_{in}$ having an angle A is rotated by an angle B, resulting in a new vector $x_{out}y_{out}$. The rotation of the vector is described in Equation(3.1) [77].

$$\begin{aligned} x_{out} &= R \cdot cos(A+B) = x_{in} \cdot cos\, B - y_{in} \cdot sin\, B \\ y_{out} &= R \cdot sin(A+B) = x_{in} \cdot sin\, B + y_{in} \cdot cos\, B, \end{aligned} \tag{3.1}$$

where R is the modulus of the vector and A is the initial angle. The rotation can also be described in a matrix form as:

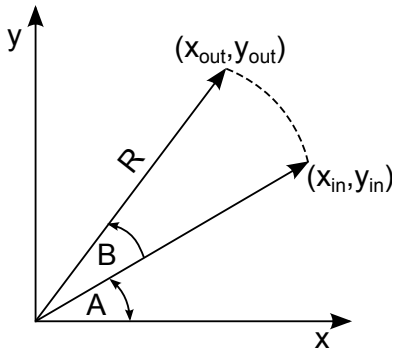


Fig. 3.2: The vector $x_{in}y_{in}$ is rotated by an angle B on the xy plane

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} cos\, B & -sin\, B \\ sin\, B & cos\, B \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \tag{3.2}$$

Instead of B as presented in Equation (3.2), a notation ϕ is applied. The basic idea dealing with the rotation-extension CORDIC computation method is described in Equation (3.3), where x_{in}, y_{in}, and ϕ are input parameters and *r* is a rotation degree. For the sake of simplicity, the proposed CORDIC equation will be derived and considered based on the circular coordinate system.

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} cos\, \phi & -sin\, \phi \\ sin\, \phi & cos\, \phi \end{bmatrix}^{r} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \tag{3.3}$$

3.3.1 Conventional CORDIC

The CORDIC algorithm performs a sequence of rotations by elementary angles. For the conventional method, the rotation ϕ on the xy plane can be decomposed into a product of *n* elementary rotations when *r=1* as:

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = cos\, \phi \begin{bmatrix} 1 & -tan\, \phi \\ tan\, \phi & 1 \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}, \tag{3.4}$$

where $cos\, \phi$ is a constant scaling factor. The elementary rotation matrix $R(\delta_i)$ describes the elementary rotation with angle $\theta_i = tan^{-1}(2^{-i})$ while i is the number of iterations. $\delta_i \in \{1, -1\}$ determines the rotation direction δ.

$$\prod_{i=0}^{n-1} R(\delta_i) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \begin{bmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{bmatrix} \tag{3.5}$$

Since the micro-rotation angle θ_i is always constant (based on a non-redundant method [82]), the constant scaling factor $cos\,\phi$ will be $\prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}}$ which approximately equals to 0.60725 with $n \longrightarrow \infty$. The maximum and minimum rotation angles are defined as $\pm \sum_{i=0}^{n-1} tan^{-1}(2^{-i})$, which is in range of [-1.74329,+1.74329]. The micro-rotation equation of the conventional CORDIC method on the circular coordinate system is presented in Equation (3.6).

$$\begin{aligned} x_{i+1} &= x_i - \delta_i \cdot 2^{-i} \cdot y_i \\ y_{i+1} &= y_i + \delta_i \cdot 2^{-i} \cdot x_i \\ z_{i+1} &= z_i - \delta_i \cdot \theta_i \end{aligned} \tag{3.6}$$

3.3.2 Double-Rotation CORDIC

The aim of the double-rotation method is to accelerate the rotation computation of the CORDIC algorithm by duplicating the elementary angle to be $2\theta_i$. Thus, the degree of parameter r in Equation (3.3) is equal to 2.

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = cos^2\,\phi \begin{bmatrix} 1 & -tan\,\phi \\ tan\,\phi & 1 \end{bmatrix}^2 \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \tag{3.7}$$

The decomposition of the production of n elementary rotations is shown in Equation (3.8) with $\theta_i = tan^{-1}(2^{-i-1})$.

$$\prod_{i=1}^{n} R(\delta_i) = \prod_{i=1}^{n} \frac{1}{(1+2^{-2i-2})} \begin{bmatrix} 1 & -\delta_i \cdot 2^{-i-1} \\ \delta_i \cdot 2^{-i-1} & 1 \end{bmatrix}^2 \tag{3.8}$$

To stabilize the scaling factor, the non-redundant number, where $\delta_i \in \{-1, 1\}$, is also applied for the double-rotation CORDIC method. Thereby, the optimal constant scaling factor is formulated by $\prod_{i=1}^{n} \frac{1}{(1+2^{-2i-2})}$ which approximately equals to 0.9219 with $n \longrightarrow \infty$. The maximum and minimum rotation angles $\pm \sum_{i=1}^{n} 2 \cdot tan^{-1}(2^{-i-1})$ are in range of [-0.9885,+0.9885]. The micro-rotation equation of the double-rotation CORDIC method on the circular coordinate system is shown in Equation (3.9).

$$\begin{aligned} x_{i+1} &= x_i - \delta_i \cdot 2^{-i} \cdot y_i - 2^{-2i-2} \cdot x_i \\ y_{i+1} &= y_i + \delta_i \cdot 2^{-i} \cdot x_i - 2^{-2i-2} \cdot y_i \\ z_{i+1} &= z_i - \delta_i \cdot 2 \cdot \theta_i \end{aligned} \tag{3.9}$$

3.3.3 Triple-Rotation CORDIC

Like the conventional and the double-rotation methods, the triple-rotation method can accelerate a micro rotation θ_i by triplicating the elementary angle θ_i to $3\theta_i$.

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = cos^3\,\phi \begin{bmatrix} 1 & -tan\,\phi \\ tan\,\phi & 1 \end{bmatrix}^3 \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \tag{3.10}$$

The decomposition of the product of *n* elementary rotation is described in Equation (3.11), where *r=3*.

$$\prod_{i=2}^{n} R(\delta_i) = \prod_{i=2}^{n+1} \frac{1}{(1+2^{-2i-4})^{\frac{3}{2}}} \begin{bmatrix} 1 & -\delta_i 2^{-i-2} \\ \delta_i 2^{-i-2} & 1 \end{bmatrix}^3 \tag{3.11}$$

The micro-rotation angle is $\theta_i = tan^i(2^{-i-2})$. The non-redundancy for rotation direction is applied and the scaling factor becomes approximately a constant value at 0.9922 with $\prod_{i=2}^{n+1} \frac{1}{(1+2^{-2i-4})^{\frac{3}{2}}}$. The maximum and minimum rotation angles can be found by $\pm \sum_{i=2}^{n+1} 3 \cdot tan^{-1}(2^{-i-2})$, which is in a range of [-0.3747, +0.3747] with $n \longrightarrow \infty$. The micro-rotation equation of the triple-rotation CORDIC method which has been simplified for *VLSI* hardware implementation is shown in Equation (3.12).

$$\begin{aligned} x_{i+1} &= x_i(1 - 2^{-2i-3} - 2^{-2i-4}) - \delta_i(2^{-i-1} + 2^{-i-2} - 2^{-3i-6})y_i \\ y_{i+1} &= \delta_i x_i(2^{-i-1} + 2^{-i-2} - 2^{-3i-6}) + (1 - 2^{-2i-3} - 2^{-2i-4})y_i \\ z_{i+1} &= z_i - \delta_i \cdot 3 \cdot \theta_i \end{aligned} \tag{3.12}$$

3.3.4 Accuracy Evaluation

Computational accuracy can be considered on either rotation mode or vectoring mode. In this dissertation, the iterative CORDIC in rotation mode on the circular coordinate system is taken into account for the accuracy verification, where parameter z_i is driven to 0 ($z_i \longrightarrow \infty$). The summation formulas of parameter z on each CORDIC method are shown as follows:

$$\begin{aligned} z_{out-CV} &= z_{in} - \textstyle\sum_{i=0}^{n-1} \delta_i \cdot tan^{-1}(2^{-i}) \\ z_{out-DR} &= z_{in} - \textstyle\sum_{i=1}^{n} \delta_i \cdot 2 \cdot tan^{-1}(2^{-i-1}) \\ z_{out-TR} &= z_{in} - \textstyle\sum_{i=2}^{n+1} \delta_i \cdot 3 \cdot tan^{-1}(2^{-i-2}) \end{aligned} \tag{3.13}$$

Tab. 3.1: Probability of rotation direction δ of the conventional, double-rotation, and triple-rotation CORDIC methods, where z_{in} is varied from 0.0 to 0.3

CORDIC method	δ		
	-1	0	1
Conventional	0.5020	0	0.4980
Proposed double-rotation	0.5051	0.1434	0.4734
Proposed triple-rotation	0.5051	0	0.4949

The conditions that the triple-rotation and double-rotation methods have more precision than the conventional method can be expressed as:

$$\begin{aligned} \left|z_{in} - \sum_{i=2}^{n+1} \delta_i \cdot 3 \cdot tan^{-1}(2^{-i-2})\right| &> \left|z_{in} - \sum_{i=1}^{n} \delta_i \cdot 2 \cdot tan^{-1}(2^{-i-1})\right| \\ &> \left|z_{in} - \sum_{i=0}^{n-1} \delta_i \cdot tan^{-1}(2^{-i})\right| \end{aligned} \tag{3.14}$$

From Equation (3.14) the rotation direction δ on each iteration depends on an intermediate parameter z_i of each method. Analysis based on mathematical assumption cannot guarantee the computational accuracy due to the non-linear equation. Therefore, the accuracy evaluation can be applied in practice by the simulation based on statistical analysis in mathematics which is *Mean Absolute Percent Error* and standard statistical indicators can be applied in practice.

The *Mean Absolute Percent Error* (*MAPE*) mathematical factor is employed for accuracy evaluation in available range [-1.74329, +1.74329], [-0.9885,+0.9885], and [-0.3747, 0.3747] of the conventional, double-rotation, and triple-rotation methods, respectively. The simulation results show that the proposed methods, double-rotation and triple-rotation, provide a smaller *MAPE* at the same number of iterations than the conventional method, which implies better accuracy as illustrated in Tab. 3.2. The *MAPE* is defined by the formula:

$$M = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{A_i - F_i}{\bar{A}} \right|, \tag{3.15}$$

where A_i is the actual value and F_i is the forecasting value which is a Matlab ideal result in this dissertation. The difference between A_i and F_i is divided by the average actual value $\bar{A}$ again. The absolute value of this calculation is summed for every forecasting points in time and divided again by the number of the fitted points n.

Four standard statistical indicators are employed for accuracy evaluation, i.e. maximum absolute error $(Max.|error|)$, minimum absolute error $(Min.|error|)$, average absolute error $(Ave.|error|)$, and standard deviation absolute error $(Std.\,Dev.|error|)$. All formulas are expressed as follows:

Tab. 3.2: The *MAPE* comparisons of x_i and y_i of the conventional, double-rotation and triple-rotation CORDIC methods, where the iteration steps *i* equal to 8, 10, and 16.

	Mean absolute percent error (MAPE)		
CORDIC	i=8	i=10	i=16
x_i, cos(ϕ)			
Conventional	0.0597%	0.0153%	2.2425E-4%
Double-rotation	0.0297%	0.0086%	1.1735E-4%
Triple-rotation	0.0225%	0.0060%	8.0427E-5%
y_i, sin(ϕ)			
Conventional	0.0481%	0.0131%	1.6301E-4%
Double-rotation	0.0232%	0.0057%	9.2696E-5%
Triple-rotation	0.0180%	0.0050%	9.3906E-5%

Maximum absolute error $(Max.|error|)$

$$Max.|error| = Max.\{|A_0 - F_0|, \cdots, |A_N - F_N|\} \tag{3.16}$$

Minimum absolute error $(Min.|error|)$

$$Min.|error| = Min.\{|A_0 - F_0|, \cdots, |A_N - F_N|\} \tag{3.17}$$

Average absolute error $(Ave.|error|)$

$$Ave.|error| = \frac{1}{N}\sum_{i=0}^{N}|A_i - F_i| \tag{3.18}$$

Standard deviation absolute error $(Std.\,Dev.|error|)$

$$\begin{aligned} X_i &= |A_i - F_i| \\ \bar{X} &= \frac{1}{N}\sum_{i=0}^{N}|A_i - F_i| \\ Std.\,Dev.|error| &= \sqrt{\frac{1}{N}\sum_{i=0}^{N}(X_i - \bar{X})^2} \end{aligned} \tag{3.19}$$

The analysis results with four statistical indicators are shown in Tabs. 3.3 and 3.4. Both tables show the comparison of the computational error of the conventional, double-rotation and triple-rotation methods based on the ideal results of Matlab/Simulink in the various number of iterations (*Ns*) at 8, 10, 16, 32, and 64. For investigation an environment, since the computational accuracy on the three parameters, i.e. x_{out}, y_{out}, and z_{out}, has to be measured, the methods will be performed on both rotation mode and vectoring mode. Then, the circular coordinate system is selected for the experimental case. Tab. 3.3

reports the analysis results of the CORDIC methods in rotating mode performed by function number 1 in Tab. 3.5. Similarly, the analysis results in vectoring mode of the CORDIC method performed by the function number 3 are illustrated in Tab. 3.4. From the tables, the double-rotation and triple-rotation methods still provide higher accuracy than the conventional method when they are analysed by the standard statistical indicators.

3.3.5 Convergence & Accuracy Trade-Off

The performance and time efficiency of the double-rotation and triple-rotation methods can be investigated and evaluated by two methods based on the computational results of the conventional method and the ideal results of Matlab/Simulink. The first method is the determination of absolute error constraint. Thereby, the absolute error is given as a different expected error ΔE of a CORDIC computational result and a Matlab/Simulink ideal result. If the actual different error Δe of a CORDIC computational result and a Matlab/Simulink ideal result is equal or less than ΔE, then the number of iterations will be kept in record. Fig. 3.3 illustrates the trade-off of the given absolute error with the number of iterations of the conventional, double-rotation, and triple-rotation methods based on Matlab/Simulink. At the same number of iterations, the triple-rotation and double-rotation methods provide higher accuracy than the conventional method. In turn, they require the smaller number of iterations at the same absolute error.

The second method is the observation of the converging parameters x, y, z of the micro-rotation. A sine-cosine function is performed by setting the initial parameters x, y, and z in rotating mode, whereas the parameter z is driven to be zero, $z \longrightarrow 0$, by the CORDIC algorithm. Based on the experiments with several values of input z, the triple-rotation method converges faster than the double-rotation and conventional methods for all three parameters. The convergence of the three parameters is captured and shown in Fig. 3.4, where z is initialized with $\phi = -0.1$ radian, y is set to 0, and x is defaulted by the constant scaling factor of each method. The proposed CORDIC methods provide better performance with a fewer number of iterations compared to the conventional. However, since the convergence range weakness of the double-rotation and triple-rotation methods, the conventional method provides better efficiency. In order to alleviate the limitation, the convergence range extension method is proposed and described in section 3.9.

Tab. 3.3: The computational accuracy analysis of the CORDIC methods in rotation mode on the circular coordinate system.

	Iteration (N)	CORDIC	Statistical Analysis			
			$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
x_{out}	8	Conventional	9.4966E-3	3.2120E-5	4.5125E-3	2.6722E-3
		Double-rotation	4.8971E-3	1.5988E-5	2.3296E-3	1.3431E-3
		Triple-rotation	3.6940E-3	2.2748E-5	1.7411E-3	1.0119E-3
	10	Conventional	2.3952E-3	4.7574E-6	1.1589E-3	6.6409E-4
		Double-rotation	1.2320E-3	6.9294E-6	5.7707E-4	3.3635E-4
		Triple-rotation	9.1174E-4	5.6572E-7	4.3358E-4	2.5278E-4
	16	Conventional	3.7365E-5	1.2253E-7	1.8090E-5	1.0543E-5
		Double-rotation	1.8925E-5	8.1072E-9	9.1003E-6	5.3043E-6
		Triple-rotation	1.4181E-5	2.6530E-8	6.7987E-6	3.9405E-6
	32	Conventional	5.8888E-10	4.6108E-13	2.7506E-10	1.6137E-10
		Double-rotation	2.9457E-10	2.9358E-12	1.3889E-10	8.0351E-11
		Triple-rotation	2.1399E-10	1.0505E-12	1.0593E-10	5.9295E-11
	64	Conventional	9.9920E-16	3.7453E-15	5.8693E-15	3.6346E-16
		Double-rotation	3.4417E-15	1.5543E-15	2.4920E-15	2.1890E-16
		Triple-rotation	1.5543E-15	1.1102E-16	7.1996e-16	2.8387E-16
y_{out}	8	Conventional	6.748E-3	1.7481E-5	2.9117E-3	1.7723E-3
		Double-rotation	3.3532E-3	8.9682E-6	1.4952E-3	8.7957E-4
		Triple-rotation	2.5376E-3	1.5211E-5	1.1163E-3	6.5960E-4
	10	Conventional	1.6862E-3	3.3777E-6	7.4364E-4	4.3493E-4
		Double-rotation	8.2529E-4	3.9484E-6	3.7026E-4	2.1735E-4
		Triple-rotation	6.1626E-4	4.1980E-7	2.7813E-4	1.6486E-4
	16	Conventional	2.6937E-5	9.8153E-8	1.1614E-5	6.8945E-6
		Double-rotation	1.3301E-5	1.6685E-8	5.8318E-6	3.4532E-6
		Triple-rotation	9.6708E-6	4.4125E-9	4.3576E-6	2.5698E-6
	32	Conventional	4.0018E-10	1.9198E-12	1.7623E-10	1.0478E-10
		Double-rotation	2.0294E-10	5.1292E-13	8.9197E-11	5.2357E-11
		Triple-rotation	1.5010E-10	2.9110E-13	6.8012E-11	3.8826E-11
	64	Conventional	5.7732E-15	4.4580E-15	4.0651E-15	5.3280E-16
		Double-rotation	2.4425E-15	2.6645E-15	9.8142E-16	5.0489E-16
		Triple-rotation	1.7764E-15	1.0012E-15	4.2967E-16	3.1539E-16

Tab. 3.4: The computational accuracy analysis of the CORDIC methods in vectoring mode on the circular coordinate system.

	Iteration (N)	CORDIC	Statistical Analysis			
			$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
x_{out}	8	Conventional	2.1886E-5	5.8874E-8	9.577E-6	9.2267E-6
		Double-rotation	6.0890E-6	3.2259E-6	3.5329E-6	2.0376E-6
		Triple-rotation	2.6975E-6	1.3162E-7	1.3317E-6	9.2651E-7
	10	Conventional	1.296E-6	8.4752E-9	5.9492E-7	5.9646E-7
		Double-rotation	3.0917E-7	6.8938E-8	1.9876E-7	1.0249E-7
		Triple-rotation	2.5691E-7	2.5071E-9	9.0108E-8	1.0678E-7
	16	Conventional	3.3458E-10	4.3225E-12	1.2284E-10	1.4963E-10
		Double-rotation	6.9108E-11	2.7643E-12	4.7638E-11	1.6298E-11
		Triple-rotation	4.5727E-11	1.3323E-14	1.9549E-11	2.3043E-11
	32	Conventional	5.5511E-15	2.2204E-15	3.3307E-16	1.3597E-16
		Double-rotation	4.1078E-15	3.2196E-16	3.7970E-15	3.6316E-16
		Triple-rotation	1.4433E-15	3.1307E-16	9.5479E-16	4.4823E-16
	64	Conventional	5.5511E-15	2.2204E-15	3.3307E-16	1.3597E-16
		Double-rotation	4.1078E-15	3.2196E-16	3.7970E-15	3.6316E-16
		Triple-rotation	1.4433E-15	3.1307E-16	9.5479E-16	4.4823E-16
z_{out}	8	Conventional	6.6160E-3	3.4315E-4	3.7572E-3	2.5093E-3
		Double-rotation	3.1040E-3	2.9878E-4	1.8719E-3	1.1285E-3
		Triple-rotation	2.3227E-3	8.0323E-4	1.5465E-3	5.8293E-4
	10	Conventional	1.6100E-3	1.1742E-4	9.3258E-4	6.3258E-4
		Double-rotation	6.7778E-4	1.0275E-4	4.2626E-4	2.6665E-4
		Triple-rotation	7.1681E-4	7.0811E-5	3.4929E-4	2.6975E-4
	16	Conventional	2.5868E-5	2.9403E-6	1.2778E-5	1.0149E-5
		Double-rotation	9.9709E-6	4.0604E-6	7.2306E-6	2.2905E-6
		Triple-rotation	9.5632E-6	1.6408E-7	4.7247E-6	4.5793E-6
	32	Conventional	3.2805E-10	1.0026E-10	2.3369E-10	9.6639E-11
		Double-rotation	1.7855E-10	7.2290E-11	9.8846E-11	4.5243E-11
		Triple-rotation	1.5460E-10	2.4420E-11	8.5503E-11	5.5706E-11
	64	Conventional	1.1102E-16	2.7756E-17	6.6613E-17	3.1646E-17
		Double-rotation	1.1102E-16	2.7756E-17	6.6613E-17	3.1646E-17
		Triple-rotation	1.1102E-16	2.7756E-17	4.1633E-17	4.3885E-17

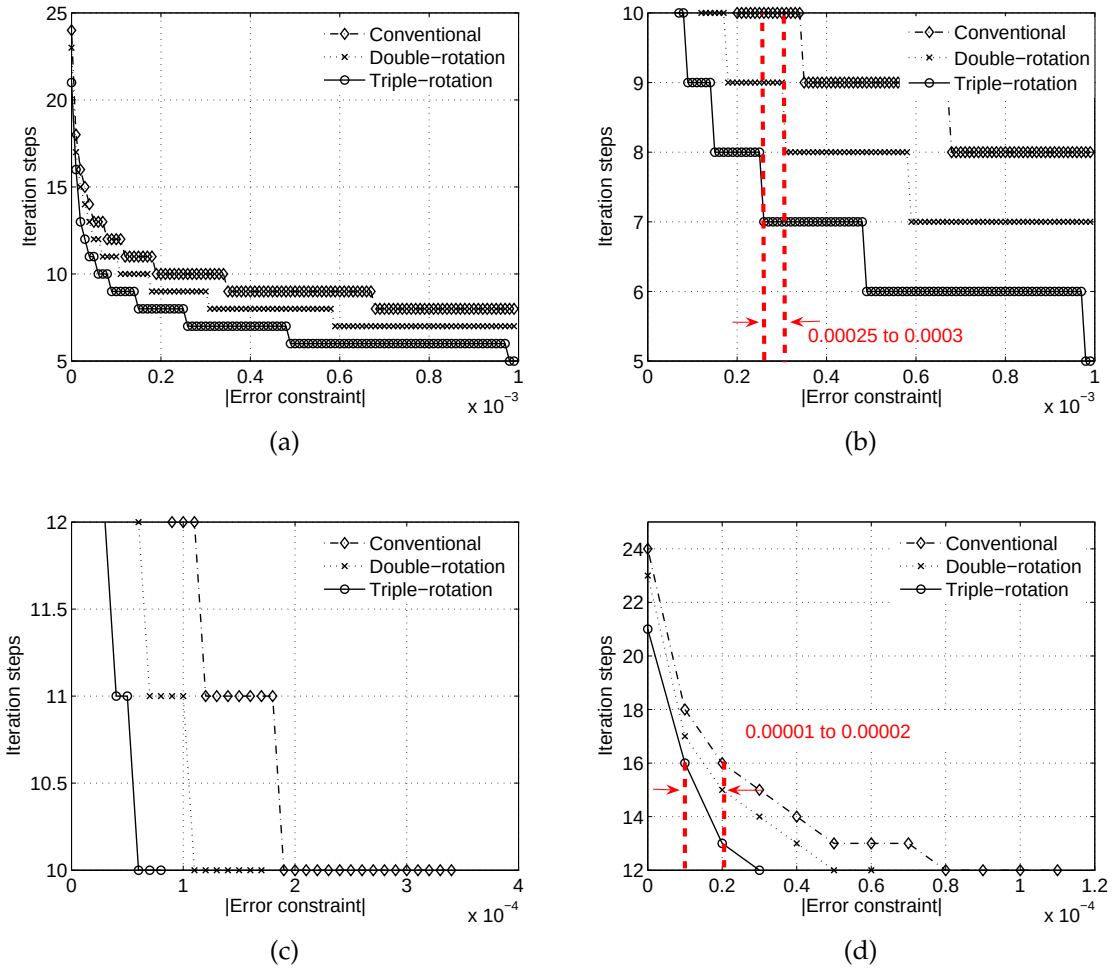
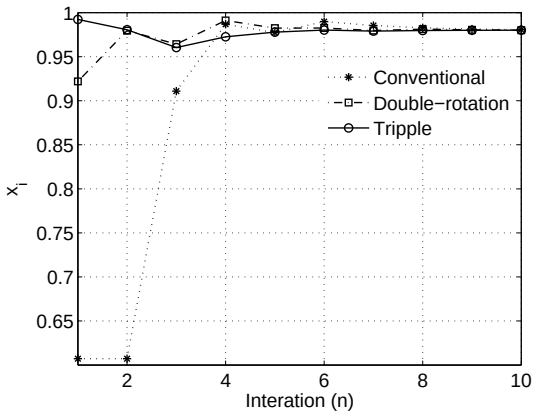
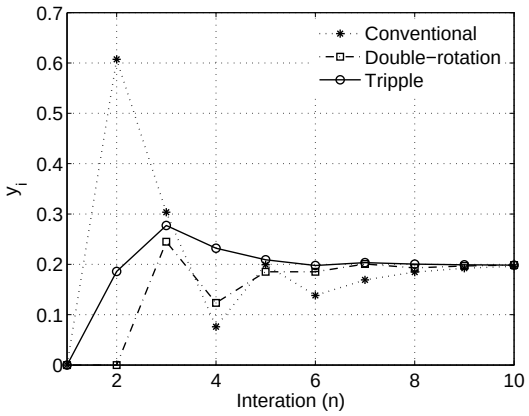


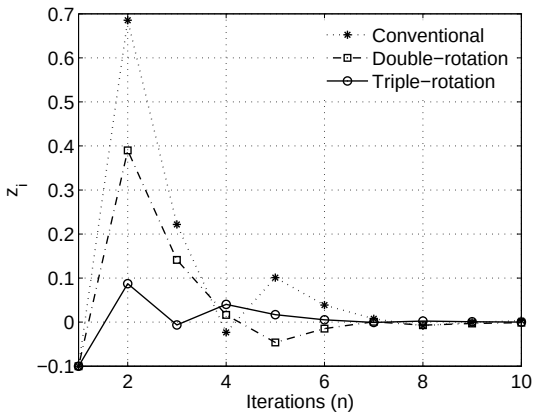
Fig. 3.3: The required iteration step when the absolute error is varied from 1.0E-8 to 1.0E-3.



(a) The convergence of parameter x



(b) The convergence of parameter y



(c) The convergence of parameter z

Fig. 3.4: The convergences of CORDIC parameters, where x_{in} is initialised with the constant scaling factors of each CORDIC, $y_{in} = 0$, and $z_{in} = \phi = -0.1$ radian.

3.4 The Circular Coordinate System

Functions in trigonometry, i.e. *Sine*, *Cosine*, *Rectangular-to-Polar* and *Polar-to-Rectangular* can be performed by the CORDIC algorithm. With the initialization of parameters (*x*, *y*, *z*), a constant scaling factor (*K*) and rotation mode, the CORDIC can build these functions efficiently. Several articles introduce applications based on the CORDIC in rotation mode. In a communication application, Michael P. Fitz [34] used CODIC's sine and cosine in the *Break-Even-Point* (*BEP*) detector which is a part of a decision processor for frequency and phase detection on a channel carrier synchronizer. B. S. Song et al. [111] implemented a general digital *FM* demodulator for *FM*, *TV*, and wireless by employing a sine decimation filter performed based on the CORDIC. T. Y. Sung et al. [112] applied this mode of the CORDIC on graphic applications for fast rendering pictures, where the sine-cosine-based functions were implemented as a hardware accelerator to support the 3-D vector interpolation algorithm. L. Vachhani et al. [119] introduced two common operations based on the CORDIC, i.e. rotating a vector in 2-D and aligning a vector on the xy plane with a specific axis, on a mobile robotic application.

The CORDIC in vectoring mode has been applied in several scientific application areas. In a synchrotron RF control application, the functions are used for phase and amplitude detection. The *LASER Inspection System* [86] was developed by *ENEA* (*Italian Agency for New Technologies, Energy and Environment*) for periodic in-vessel inspection in large fusion machines. The main purpose of this system is to obtain a complete 3-D mapping of the in-vessel surface where amplitude and phase components of image will be modulated with beam signal and return to a receiver module. A demodulation module on the receiver, which was used to split the two components was designed and implemented base on the CORDIC. The closed-loop RF control of ion beams in synchrotron [59] [42] [139] applied the function *Polar-to-Rectangular* for a phase detector implementation. The phase detector observes a different phase value of the *"Beam Signal"* and the *"Gap Signal"*. The phase difference will then be filtered by a filter module and the filtered phase difference will be fed to the controller for processing. Afterwards, the processing result will be applied to the damp beam oscillation of beam signal. S. W. Lee et al. [65] applied the *Cartesian-to-Polar* for chip implementation of a linear interpolation system.

There are two rotation modes for the micro-rotation of the double-rotation and triple-rotation methods on the circular coordinate system, which are rotation mode and vectoring mode, as shown in Equations (3.20) and (3.21).

The micro-rotation of double-rotation method on the circular coordinate system

$$\begin{aligned} x_{i+1} &= x_i - \delta_i \cdot 2^{-i} \cdot y_i - 2^{-2i-2} \cdot x_i \\ y_{i+1} &= y_i + \delta_i \cdot 2^{-i} \cdot x_i - 2^{-2i-2} \cdot y_i \\ z_{i+1} &= z_i - \delta_i \cdot 2 \cdot tan^{-1}(2^{-i-1}) \end{aligned} \tag{3.20}$$

Tab. 3.5: Elementary functions with initial parameters in rotation mode and vectoring mode on the circular coordinate system of the CORDIC.

No.	Functions	δ_i	Initial parameters		
			x_{in}	y_{in}	z_{in}
1	$x_{out} = cos(z_{in})$ $y_{out} = sin(z_{in})$	Sign(z_i)	K	0	given value
2	$x_{out} = K \cdot (x_{in} \cdot cos(z_{in}) - y_{in} \cdot sin(z_{in}))$ $y_{out} = K \cdot (x_{in} \cdot sin(z_{in}) + y_{in} \cdot cos(z_{in}))$		given value	given value	given value
3	$x_{out} = K \cdot \sqrt{x_{in}^2 + y_{in}^2}$ $z_{out} = z_{in} + tan^{-1}\left(\frac{y_{in}}{x_{in}}\right)$	-Sign(y_i)	given value	given value	given value

The micro-rotation of triple-rotation method on the circular coordinate system

$$\begin{aligned} x_{i+1} &= x_i \cdot (1 - 2^{-2i-3} - 2^{-2i-4}) - \delta_i(2^{-i-1} + 2^{-i-2} - 2^{-3i-6}) \cdot y_i \\ y_{i+1} &= \delta_i \cdot x_i \cdot (2^{-i-1} + 2^{-i-2} - 2^{-3i-6}) + (1 - 2^{-2i-3} - 2^{-2i-4}) \cdot y_i \\ z_{i+1} &= z_i - \delta_i \cdot (3 \cdot tan^{-1}(2^{-i-2})) \end{aligned} \tag{3.21}$$

Rotation mode, $z \longrightarrow 0$

$$\delta_i = \begin{cases} -1 & for\, z_i < 0, \\ 1 & otherwise. \end{cases} \tag{3.22}$$

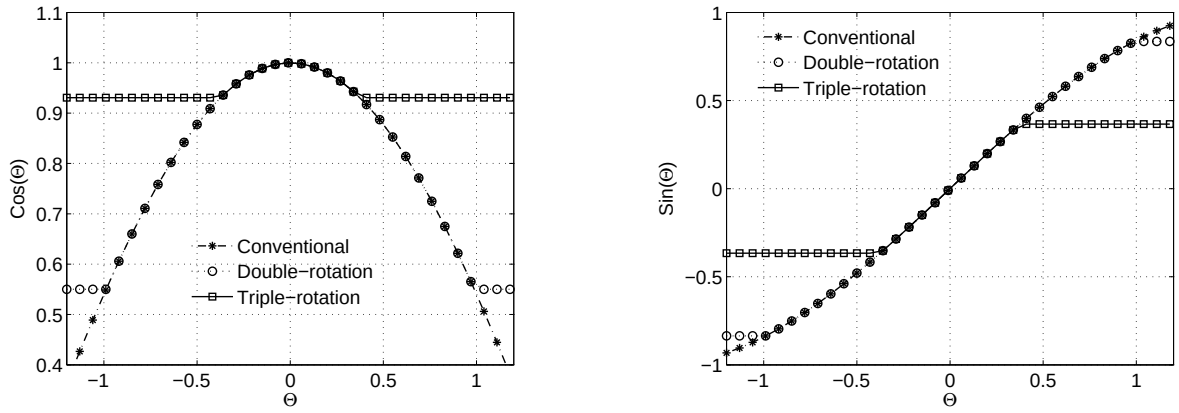
Vectoring mode, $y \longrightarrow 0$

$$\delta_i = \begin{cases} 1 & for\, y_i < 0, \\ -1 & otherwise. \end{cases} \tag{3.23}$$

The rotation mode is enabled when parameter z is converged to 0, $z \longrightarrow 0$. If parameter y is converged to 0, $y \longrightarrow 0$, then the micro-rotation will run in vectoring mode. Equations (3.22) and (3.23) illustrate the rotation direction δ_i in rotation mode and vectoring mode. Tab. 3.5 shows the elementary functions performed in the two modes of the CORDIC.

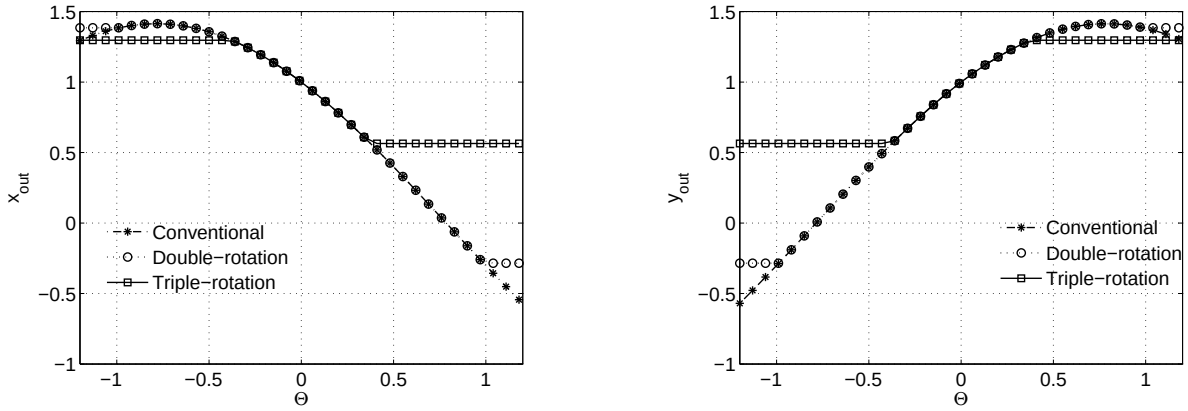
3.4.1 Convergence

In Tab. 3.5, the function numbers 1 and 2 in rotation mode and the function number 3 in vectoring mode are analysed and modelled on Matlab/Simulink. Convergence ranges of the three elementary functions are considered and compared based on the conventional, double-rotation, and triple-rotation methods as shown in Figs. 3.5 to 3.7.



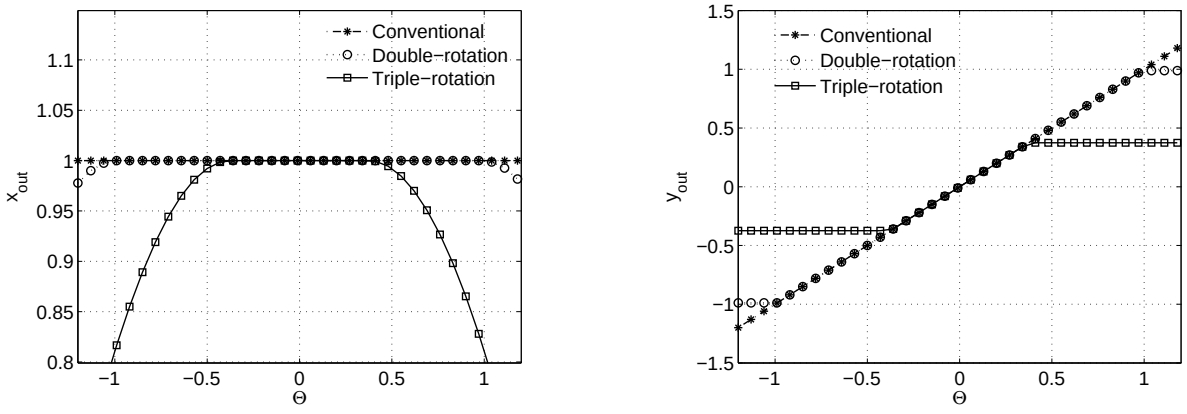
(a) Cos(Θ) (b) Sin(Θ)

Fig. 3.5: Convergence range of cosine and sine functions performed by the three CORDIC methods in rotation mode $z_i \longrightarrow 0$ with $\Theta = z_{in}$.



(a) $(x_{in} \cdot cos(z_{in}) - y_{in} \cdot sin(z_{in}))$ (b) $(y_{in} \cdot sin(z_{in}) + x_{in} \cdot cos(z_{in})))$

Fig. 3.6: Reformulation of function number 2 consists of $(x_{in} \cdot cos(z_{in}) - y_{in} \cdot sin(z_{in}))$ and $(y_{in} \cdot sin(z_{in}) + x_{in} \cdot cos(z_{in}))$ with the convergence range from -1 to 1 radian with the three CORDIC methods.



(a) $\sqrt{x_{in}^2 + y_{in}^2}$ (b) $z_{in} + tan^{-1}\frac{y_{in}}{x_{in}}$

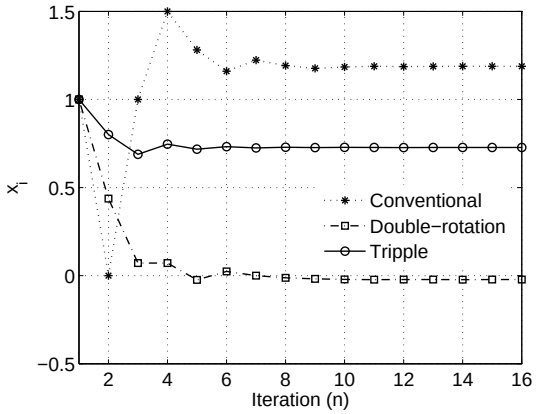
Fig. 3.7: Functions $\sqrt{x_{in}^2 + y_{in}^2}$, $z_{in} + tan^{-1}\frac{y_{in}}{x_{in}}$ with the convergence range from -1 to 1 radian which are performed by the three CORDIC methods.

The function numbers 1, 2, and 3 consist of two basic trigonometric functions. With Matlab/Simulink, the function number 1 is modeled and simulated based on the conventional, double-rotation, and triple-rotation methods. The simulation results, where x_{in} is initialized by the constant scaling factors *K*s of each CORDIC method, i.e. 0.6725, 0.9219 and 0.9922, are plotted and presented in Fig. 3.5. For simplicity, the function number 2 is multiplied by the inversion of the constant scaling factors for simple comparison, i.e. 1.64676, 1.08473 and 1.00783. Afterwards, its reformulation becomes $x_{out} = (x_{in} \cdot cos(z_{in}) - y_{in} \cdot sin(z_{in}))$ and $y_{out} = (x_{in} \cdot sin(z_{in}) + y_{in} \cdot cos(z_{in}))$. Then the simulation results are shown in Fig 3.6.

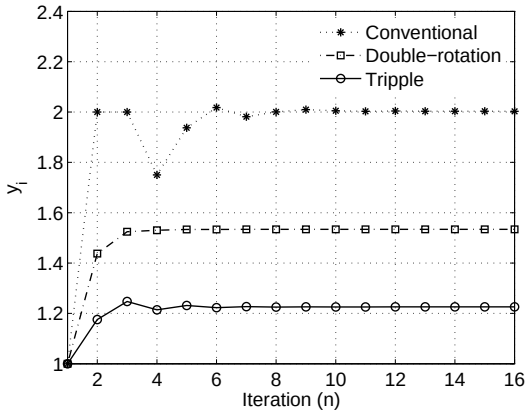
Likewise, the function number 3 is simplified when x_{out} are multiplied by the inversion of the constant scaling factors. The simulation results can be depicted in Fig. 3.7. From these simulation results in Figs. 3.5 to 3.7, the conventional, double-rotation, and triple-rotation methods provide convergence range of [-1.74329,+1.74329], [-0.9885,+0.9885] and [-0.3747,+0.3747], respectively. Figs. 3.8 and 3.9 exhibit the convergence of parameters x_i, y_i, and z_i in rotation mode and vectoring mode. From these figures, the parameters of the double-rotation and triple-rotation methods provide the convergent response faster than the conventional method.

3.4.2 Accuracy

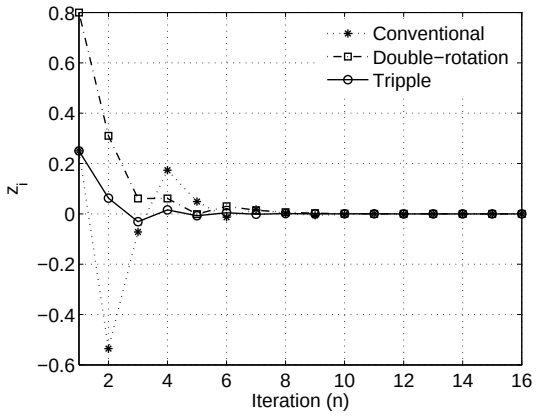
The *MAPE* on Equation (3.15) is applied to measure the accuracy of rotation mode and vectoring mode as illustrated in Tab. 3.6. The convergence range applys for the investigation based on the convergence range of the triple-rotation method at [-0.3747,+0.3747]. The double-rotation and triple-rotation methods provide better accuracy significantly than the conventional method when the number of iterations N is at 8, 10, and 16 as reported in Tab. 3.6.



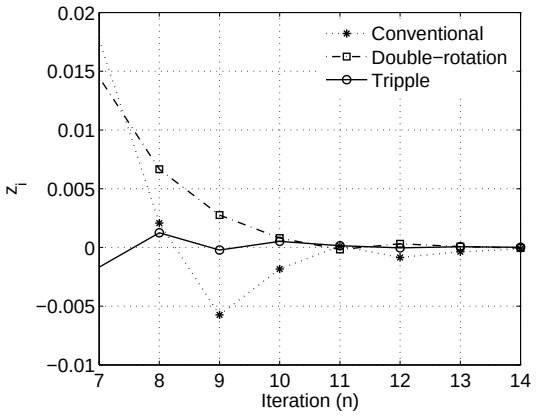
(a) Convergence of x_i



(b) Convergence of y_i

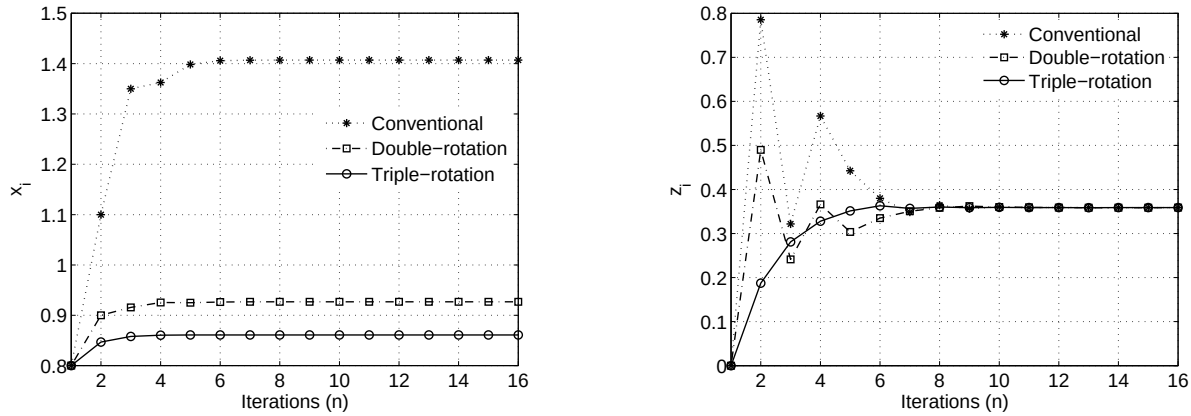


(c) Convergence of z_i



(d) Zoomed convergence of z_i

Fig. 3.8: Convergence parameters x_i, y_i, z_i of functions $K_c(x_{in} \cdot cos(z_{in}) - y_{in} \cdot sin(z_{in}))$, $K_c(x_{in} \cdot sin(z_{in}) + y_{in} \cdot cos(z_{in}))$ performed by the conventional, double-rotation, and triple-rotation methods, where z_{in} = Θ= 0.25 radian and x_{in} and y_{in} =1.



(a) Convergence of x_i

(b) Convergence of z_i

(c) Convergence of y_i

(d) Zoomed convergence of y_i

Fig. 3.9: Convergence parameters x_i, y_i, z_i of functions $x_{out} = K_c\sqrt{x_{in}^2 + y_{in}^2}$ and $z_{out} = z_{in} + tan^{-1}\frac{y_{in}}{x_{in}}$ performed by the conventional, double-rotation, and triple-rotation methods, where x_{in} = 0.8, y_{in}=0.3, and z_{in} =0.

Tab. 3.6: The *MAPE* comparisons of x_i, y_i, and z_i of the conventional, double-rotation and triple-rotation methods, where the number of iterations *N* is varied from 8 to 64.

CORDIC		*Mean absolute percent error (MAPE)*				
		N=8	*N=10*	*N=16*	*N=32*	*n=64*
Rotation mode						
x_i	Conventional	0.6128%	0.1333%	1.8247E-3%	3.0364E-8%	4.6064E-14%
	Double-rotation	0.2743%	6.5424E-2%	1.0466E-3%	1.4453E-8%	3.2458E-13%
	Triple-rotation	0.2275%	5.0472E-2%	8.4154E-4%	1.0425E-8%	6.5581E-14%
y_i	Conventional	0.3022%	5.8196E-2%	9.2513E-4%	1.8024E-8%	3.1147E-14%
	Double-rotation	0.1351%	3.1805E-2%	5.4413E-4%	7.7169E-9%	3.7551E-13%
	Triple-rotation	0.1113%	2.9384E-2%	4.5119E-4%	5.2211E-9%	1.8319E-14%
Vectoring mode						
x_i	Conventional	9.577E-4%	5.9492E-5%	1.2284E-8%	3.3307E-14%	3.3307E-14%
	Double-rotation	3.5329E-4%	1.9876E-5%	4.7638E-9%	3.7970E-13%	3.7970E-13%
	Triple-rotation	1.3317E-4%	9.0108E-6%	2.4656E-9%	9.5479E-14%	9.5479E-14%
z_i	Conventional	1.8278%	0.6547%	7.2321E-3%	1.2154E-7%	3.9413E-14%
	Double-rotation	1.2314%	0.2249%	4.1134E-3%	5.7973E-8%	3.9413-14%
	Triple-rotation	0.95799%	0.1553%	2.5730E-3%	4.6422E-8%	1.8319E-14%

Tabs. 3.3 and 3.4 depict the statistical analysis of the functions performed by the CORDIC methods in rotation mode and vectoring mode on the circular coordinate system. The four statistical indicators, i.e. maximum absolute error $(Max.|error|)$, minimum absolute error $(Min.|error|)$, average absolute error $(Ave.|error|)$, and standard deviation absolute error $(Std.\,Dev.|error|)$, in Equations (3.16) to (3.19) are employed. These indicators are applied, where the number of iterations *N* is set to 8, 10, 16, 32, and 64, respectively. From Tabs. 3.3 and 3.4 the double-rotation and triple-rotation methods provide better computational accuracy than the conventional on both rotation and vectoring modes at the same number of iterations

3.5 The Hyperbolic Coordinate System

The hyperbolic elementary functions such as *sinh()*, *cosinh()* can be performed by the CORDIC algorithm on the hyperbolic coordinate system. By changing initial parameters, a constant scaling factor and rotation modes, the functions can be accomplished as illustrated in Tab. 3.7. The function numbers 1, 2 and 3 can be carried out in rotation mode and vectoring mode. There are a few applications in modern scientific, where the hyperbolic functions are normally used to solve the non-linear problem. For example, the modern adaptive filtering algorithm [19] employs the hyperbolic functions for tuning adaptive gains at runtime. The algorithm explains an error function (a feedback error) in a context

of a sine hyperbolic form, $e(n) = d(n) - \sum_{k=0}^{N-1} sinh\theta_k \cdot x(n-k)$. Nouri et al. [88] then proposed the adaptive filtering algorithm based on the hyperbolic function, where adaptive gains (ωs) were computed by the cosine hyperbolic, $\omega_k = \frac{cosh(\alpha\sqrt{1-(\frac{2\cdot n}{N-1})^2})}{cosh(\alpha)}$, $|n| \leq \frac{N-1}{2}$. In addition, the hyperbolic functions are also used to perform basic mathematic functions such as square-root and natural logarithm which will be discussed in detail in section 3.8.

The micro-rotation of the double-rotation and triple-rotation methods based on non-redundant CORDIC are explained in Equations (3.24) and (3.25).

The micro-rotation of double-rotation method on the hyperbolic coordinate system

$$\begin{aligned} x_{i+1} &= x_i + \delta_i \cdot 2^{-i} \cdot y_i + 2^{-2i-2} \cdot x_i \\ y_{i+1} &= y_i + \delta_i \cdot 2^{-i} \cdot x_i + 2^{-2i-2} \cdot y_i \\ z_{i+1} &= z_i - \delta_i \cdot 2 \cdot tanh^{-1}(2^{-i-1}) \end{aligned} \tag{3.24}$$

The micro-rotation of triple-rotation method on the hyperbolic coordinate system

$$\begin{aligned} x_{i+1} &= x_i \cdot (1 + 2^{-2i-3} + 2^{-2i-4}) + \delta_i(2^{-i-1} + 2^{-i-2} + 2^{-3i-6}) \cdot y_i \\ y_{i+1} &= \delta_i \cdot x_i \cdot (2^{-i-1} + 2^{-i-2} + 2^{-3i-6}) + (1 + 2^{-2i-3} + 2^{-2i-4}) \cdot y_i \\ z_{i+1} &= z_i - \delta_i \cdot (3 \cdot tanh^{-1}(2^{-i-2})) \end{aligned} \tag{3.25}$$

Rotation mode, $z \longrightarrow 0$

$$\delta_i = \begin{cases} -1 & for\, z_i < 0, \\ 1 & otherwise. \end{cases} \tag{3.26}$$

Vectoring mode, $y \longrightarrow 0$

$$\delta_i = \begin{cases} 1 & for\, y_i < 0, \\ -1 & otherwise. \end{cases} \tag{3.27}$$

Likewise the circular coordinate system, the CORDIC in rotation mode and vectoring mode on the hyperbolic coordinate system can be enabled by driving parameters either z or y to zero, $z \longrightarrow 0$ or $y \longrightarrow 0$. Equations 3.26 and 3.27 show the rotation direction δ_i in rotation mode and vectoring mode.

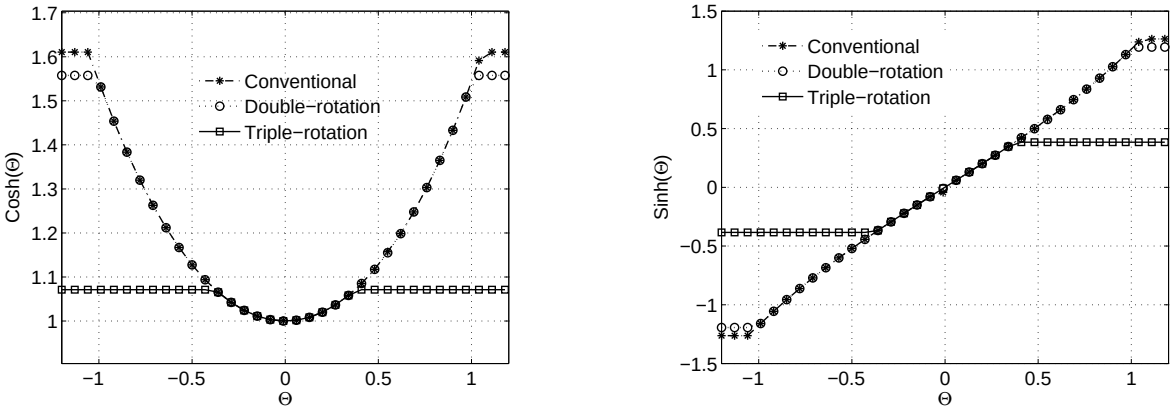
3.5.1 Convergence

In Tab. 3.7, the function numbers 1 and 2 in rotation mode and the function number 3 in vectoring mode are analysed and modelled on Matlab/Simulink. The convergence ranges of the elementary functions performed by the conventional, double-rotation, and triple-rotation CORDIC methods are compared and shown in Figs. 3.10 to 3.12.

The function numbers 1, 2, and 3 consist of two basic hyperbolic functions. The function number 1 is modelled and simulated based on the conventional, double-rotation,

Tab. 3.7: The elementary functions with initial parameters in rotation mode and vectoring mode on the hyperbolic coordinate system of CORDIC.

No.	Functions	δ_i	Initial parameters		
			x_{in}	y_{in}	z_{in}
1	$x_{out} = cosh(z_{in})$ $y_{out} = sinh(z_{in})$	$Sign(z_i)$	K^{-1}	0	given value
2	$x_{out} = K^{-1} \cdot (x_{in} \cdot cosh(z_{in}) + y_{in} \cdot sinh(z_{in}))$ $y_{out} = K^{-1} \cdot (y_{in} \cdot sinh(z_{in}) + x_{in} \cdot cosh(z_{in}))$		given value	given value	given value
3	$x_{out} = K^{-1} \cdot \sqrt{x_{in}^2 - y_{in}^2}$ $z_{out} = z_{in} + tanh^{-1}\left(\frac{y_{in}}{x_{in}}\right)$	$-Sign(y_i)$	given value	given value	given value

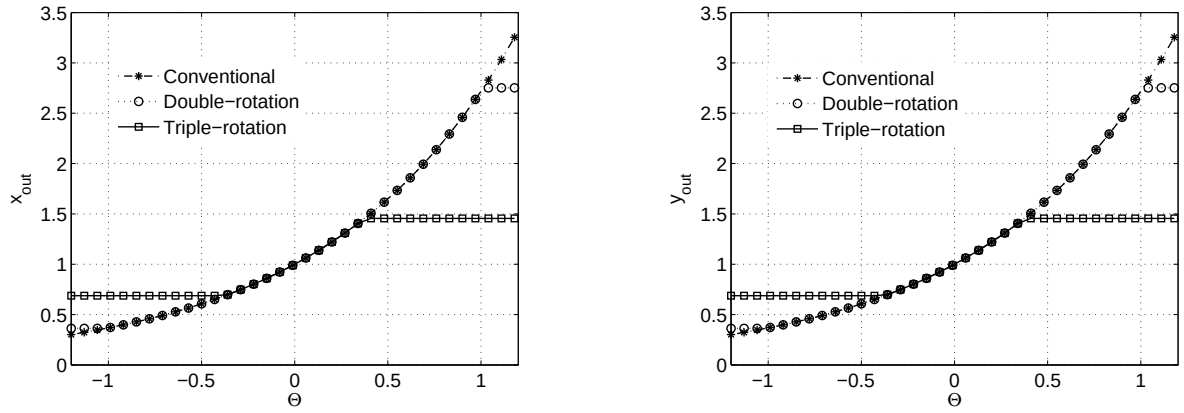


(a) Cosh(Θ) (b) Sinh(Θ)

Fig. 3.10: Available ranges of hyperbolic cosine and sine functions performed by the conventional, double-rotation, and triple-rotation methods in rotation mode $z_i \longrightarrow 0$.

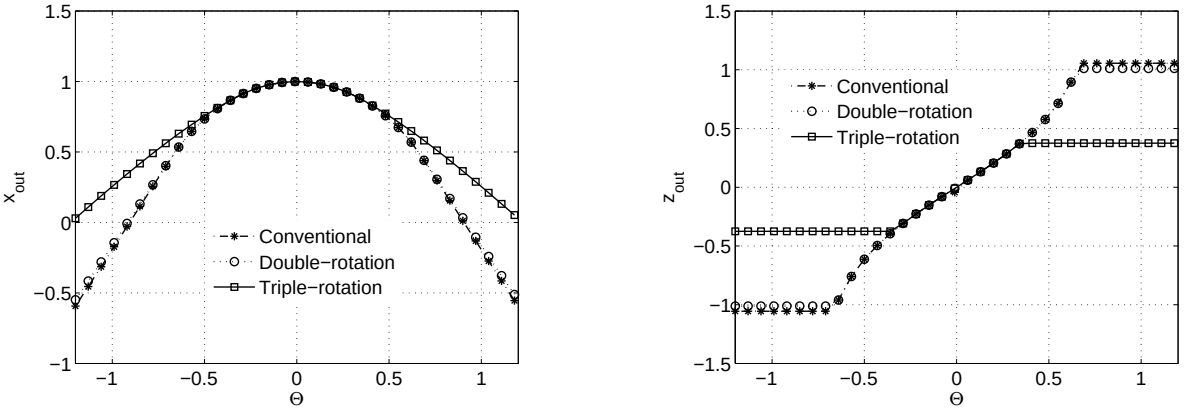
and triple-rotation methods on Matlab/Simulink. The simulation results, where x_{in} is initialized by the constant scaling factors *K*s, i.e. 1.2051, 1.0893, and 1.0079, are plotted and presented in Fig. 3.10. For simplicity, the function number 2 is also multiplied by the inversion of the constant scaling factors for comparison, i.e. 1.64676, 1.08473, and 1.00783. Afterwards, its reformulation becomes $x_{out} = (x_{in} \cdot cosh(z_{in}) + y_{in} \cdot sinh(z_{in}))$ and $y_{out} = (x_{in} \cdot sinh(z_{in}) + y_{in} \cdot cosh(z_{in}))$. The simulation results are illustrated in Fig. 3.11.

Similarly, the function number 3 is simplified when the x_{out} is multiplied by the inversion of the constant scaling factors, and the simulation results are depicted in Fig. 3.12. Based on the simulation results from Figs. 3.10 to 3.12, the conventional, double-rotation, and triple-rotation methods provide convergence ranges Θs at [-1.74329,+1.74329], [-0.9885,+0.9885], and [-0.3747,+0.3747], respectively. Figs. 3.13 and 3.14 exhibit the convergence of parameters x_i, y_i, and z_i in rotation mode and vectoring mode. From the figures, the parameters of the double-rotation and triple-rotation methods provide the convergent response faster



(a) $K_c(x_{in} \cdot cosh(z_{in}) + y_{in} \cdot sinh(z_{in}))$ (b) $K_c(y_{in} \cdot sinh(z_{in}) + x_{in} \cdot cosh(z_{in})))$

Fig. 3.11: Functions $K_c(x_{in} \cdot cosh(z_{in}) + y_{in} \cdot sinh(z_{in}))$, $K_c(y_{in} \cdot sinh(z_{in}) + x_{in} \cdot cosh(z_{in}))$ with available ranges from -1 to 1 radian performed by the double-rotation and triple-rotation methods compared to the conventional method at output x_n and y_n.



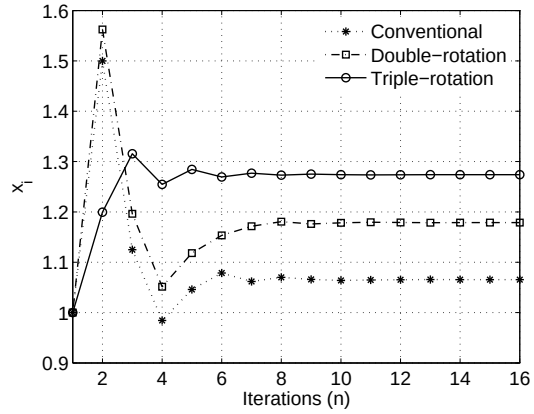
(a) $K_c^{-1}\sqrt{x_{in}^2 - y_{in}^2}$ (b) $z_{in} + tanh^{-1}\frac{y_{in}}{x_{in}}$

Fig. 3.12: Functions $K_c^{-1}\sqrt{x_{in}^2 - y_{in}^2}$, $z_{in} + tanh^{-1}\frac{y_{in}}{x_{in}}$ with available ranges from -1 to 1 radian performed by the double-rotation and triple-rotation methods compared to the conventional method at outputs x_n and y_n.

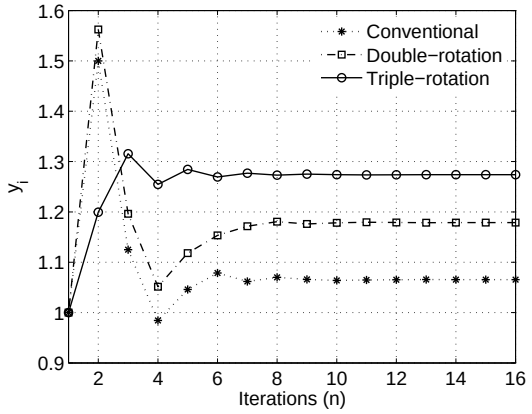
than the conventional method.

3.5.2 Accuracy

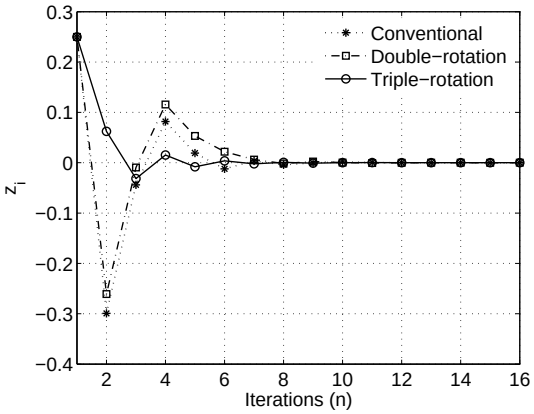
The *MAPE* on Equation (3.15) is applied to measure the functional accuracy in rotation mode and vectoring mode as illustrated in Tab. 3.8. The convergence range applied for the investigation based on the convergence range of the triple-rotation CORDIC algorithm of [-0.3747,+0.3747]. The double-rotation and triple-rotation CORDIC methods significantly provide better accuracy than the conventional method when the number of iterations N



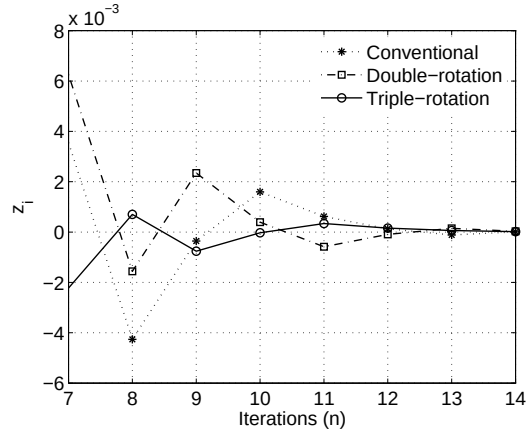
(a) Convergence of x_i



(b) Convergence of y_i



(c) Convergence of z_i

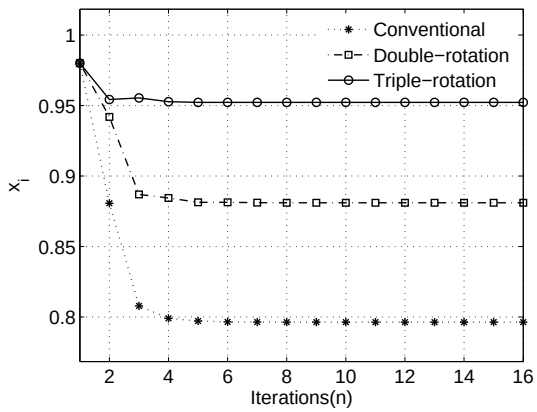


(d) Zoomed convergence of z_i

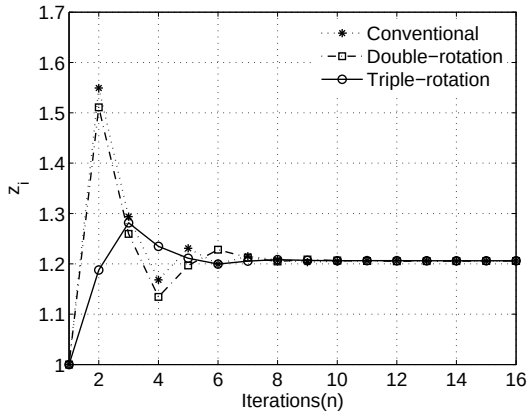
Fig. 3.13: Convergence parameters x_i, y_i, z_i of functions $K_c(x_{in} \cdot cosh(z_{in}) - y_{in} \cdot sinh(z_{in}))$, $K_c(x_{in} \cdot sinh(z_{in}) + y_{in} \cdot cosh(z_{in}))$ performed by the three CORDIC methods, where z_{in} = Θ= 0.25 radian and x_{in} and y_{in} =1.

is in between 8 and 16 iterations as shown in Tab 3.8.

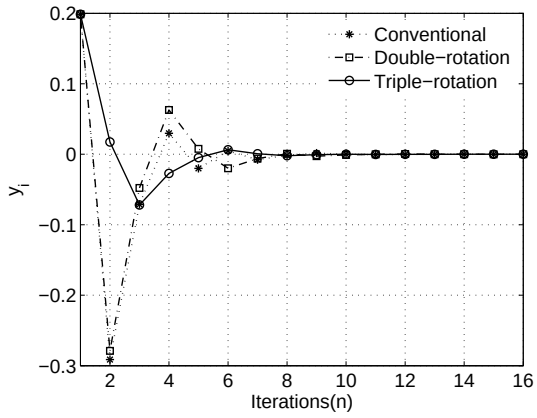
Tabs. 3.9 and 3.10 depict the statistical analysis of the functions performed by CORDIC methods in rotation mode and vectoring mode on the hyperbolic coordinate system. The four statical indicators, i.e. maximum absolute error $(Max.|error|)$, minimum absolute error $(Min.|error|)$, average absolute error $(Ave.|error|)$, and standard deviation absolute error $(Std.\,Dev.|error|)$, as shown in Equations 3.16-3.19 are applied, where the number of iterations *N* is set to 8, 10, 16, 32, and 64, respectively. From Tabs. 3.9 and 3.10, the double-rotation and triple-rotation methods provide better computational accuracy than the conventional method both rotation mode and vectoring mode at the same number of iterations.



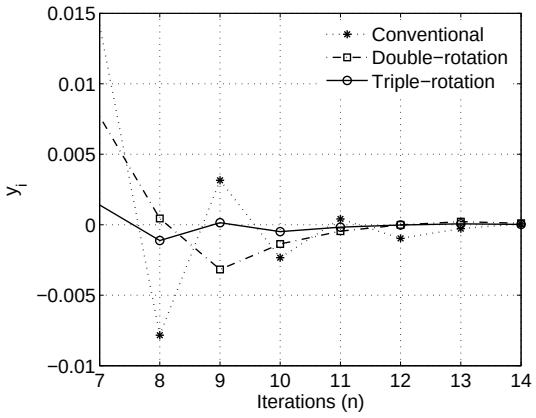
(a) Convergence of x_i



(b) Convergence of z_i



(c) Convergence of y_i



(d) Zoomed convergence of y_i

Fig. 3.14: Convergence parameters x_i, y_i, z_i of functions $K_c^{-1}\sqrt{x_{in}^2 - y_{in}^2}$, $z_{in} + tanh^{-1}\frac{y_{in}}{x_{in}}$ performed by the three CORDIC methods, where x_{in} = 0.5, y_{in}=0.3, and z_{in} =0.

Tab. 3.8: The *MAPE* comparisons of x_i and y_i of the conventional, double-rotation and triple-rotation CORDIC methods in the hyperbolic coordinate system, where the iteration step N is varied from 8 to 64.

CORDIC		*Mean absolute percent error (MAPE)*				
		N=8	*N=10*	*N=16*	*N=32*	*N=64*
Rotation mode						
x_i	Conventional	3.6523E-2%	1.0388E-2%	1.3819E-4%	2.4639E-9%	4.5402E-14%
	Double-rotation	3.9908E-2%	8.8879E-3%	1.5301E-4%	5.3909E-6%	5.3912E-6%
	Triple-rotation	2.9041E-2%	7.4149E-3%	1.2449E-4%	2.1216E-9%	4.4438E-14%
y_i	Conventional	1.0377%	0.2820%	4.3504E-3%	7.2197E-8%	3.7459E-14%
	Double-rotation	1.1063%	0.2488%	4.4642E-3%	5.3977E-6%	5.3912E-6%
	Triple-rotation	0.8584%	0.2208%	3.2056E-3%	6.1376E-8%	4.7295E-14%
Vectoring mode						
x_i	Conventional	2.5935E-4%	1.5246E-5%	4.0057E-6%	5.600E-6%	5.6000E-6%
	Double-rotation	3.9121E-4%	2.8152E-5%	5.3973E-6%	5.3912E-6%	5.3912E-6%
	Triple-rotation	2.3360E-4%	1.1304E-5%	3.1102E-6%	4.1102E-6%	4.1102E-6%
z_i	Conventional	0.1669%	3.8573E-2%	6.7091E-4%	1.0597E-8%	2.0230E-14%
	Double-rotation	0.1685%	0.0382%	6.8107E-4%	1.0699E-8%	2.2150E-12%
	Triple-rotation	9.7178E-2%	3.0303E-2%	3.7467E-4%	8.6239E-9%	1.9156E-14%

Tab. 3.9: The computational accuracy analysis of the three CORDIC methods in rotation mode on the hyperbolic coordinate system.

	Iteration (N)	CORDIC	Statistical Analysis			
			$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
x_{out}	8	Conventional	1.0454E-3	4.4976E-6	3.7437E-4	2.5993E-4
		Double-rotation	9.4328E-4	3.7396E-5	4.0953E-4	2.7887E-4
		Triple-rotation	7.5076E-4	5.3817E-7	2.9775E-4	2.0272E-4
	10	Conventional	2.3763E-4	1.0131E-5	1.0667E-4	6.9132E-5
		Double-rotation	2.0879E-4	5.9261E-6	9.1019E-5	6.1567E-5
		Triple-rotation	1.9132E-4	6.0864E-6	7.6019E-5	5.0738E-5
	16	Conventional	3.6330E-6	3.6972E-8	1.4165E-6	1.0519E-6
		Double-rotation	3.4724E-6	2.1847E-7	1.5688E-6	1.1026E-6
		Triple-rotation	3.2722E-6	2.4003E-8	1.2775E-6	8.8956E-7
	32	Conventional	6.5357E-11	2.8255E-12	2.5289E-11	1.7650E-11
		Double-rotation	5.6332E-8	5.4180E-8	5.5092E-8	6.7317E-10
		Triple-rotation	4.6612E-11	4.1172E-12	2.1747E-11	1.1073E-11
	64	Conventional	1.3323E-15	0	4.6524E-16	2.8870E-16
		Double-rotation	5.6356E-8	5.4182E-8	5.5095E-8	6.8051E-10
		Triple-rotation	1.3323E-15	0	4.5466E-16	3.0182E-16
y_{out}	8	Conventional	3.939E-3	2.5254E-5	1.8994E-3	1.1492E-3
		Double-rotation	3.9421E-3	1.6432E-4	2.0453E-3	1.1718E-3
		Triple-rotation	2.8342E-3	2.4856E-6	1.527E-3	8.9864E-4
	10	Conventional	9.6709E-4	6.8359E-5	5.2568E-4	2.7433E-4
		Double-rotation	9.7814E-4	3.0864E-5	4.6286E-4	2.927E-4
		Triple-rotation	7.4779E-4	2.8589E-5	3.9104E-4	2.2185E-4
	16	Conventional	1.5270E-5	2.1958E-7	7.4393E-6	5.0062E-6
		Double-rotation	1.5314E-5	9.5075E-7	7.9895E-6	4.8151E-6
		Triple-rotation	1.1465E-5	8.7955E-8	6.2389E-6	3.5918E-6
	32	Conventional	2.2531E-10	2.0318E-11	1.2835E-10	7.4526E-11
		Double-rotation	1.6333E-8	5.3805E-9	1.0873E-8	3.3783E-9
		Triple-rotation	1.7681E-10	2.4451E-11	1.1097E-10	4.2634E-11
	64	Conventional	1.9429E-16	0	7.4676E-17	5.6623E-17
		Double-rotation	1.6417E-8	5.4002E-9	1.0874E-8	3.4161E-9
		Triple-rotation	2.7756E-16	0	9.9127E-17	6.5283E-17

Tab. 3.10: The computational accuracy analysis of the three CORDIC methods in vectoring mode on the hyperbolic coordinate system.

	Iteration (N)	CORDIC	Statistical Analysis			
			$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
x_{out}	8	Conventional	7.0665E-6	1.4921E-7	2.4815E-6	2.4538E-6
		Double-rotation	8.8263E-6	1.2682E-6	3.7486E-6	2.3517E-6
		Triple-rotation	2.3844E-5	1.9876E-5	2.2314E-5	1.0355E-6
	10	Conventional	4.0093E-7	1.6618E-10	1.4527E-7	1.4632E-7
		Double-rotation	5.9782E-7	1.2841E-7	2.7021E-7	1.5356E-7
		Triple-rotation	2.4163E-5	2.2049E-5	2.3252E-5	6.5856E-7
	16	Conventional	1.0251E-10	6.8978E-13	3.8428E-11	6.2234E-7
		Double-rotation	5.3450E-8	4.9012E-8	5.1592E-8	1.381E-9
		Triple-rotation	2.4169E-5	2.2180E-5	2.3337E-5	6.2234E-7
	32	Conventional	5.5511E-16	0	1.5332E-16	1.5091e-016
		Double-rotation	5.3372E-8	4.8978E-8	5.1533E-8	1.3743E-9
		Triple-rotation	2.4169E-5	2.2180E-5	2.3337E-5	6.2234E-7
	64	Conventional	5.5511E-16	0	1.5332E-16	1.5091E-16
		Double-rotation	5.3372E-8	4.8978E-8	5.1533E-8	1.3743E-9
		Triple-rotation	2.4169E-5	2.2180E-5	2.3337E-5	6.2234E-7
z_{out}	8	Conventional	3.8360E-3	5.6011E-4	2.0107E-3	1.0960E-3
		Double-rotation	3.901E-3	5.9129E-5	2.0231E-3	1.0651E-3
		Triple-rotation	2.7216E-3	1.0490E-4	1.1408E-3	9.202E-4
	10	Conventional	9.3949E-4	1.8836E-5	4.6652E-4	2.0985E-4
		Double-rotation	9.7136E-4	8.9001E-5	4.5536E-4	3.0554E-4
		Triple-rotation	7.1907E-4	3.7514E-5	3.7108E-4	2.0985E-4
	16	Conventional	1.4835E-5	1.1817E-6	8.0412E-6	4.0281E-6
		Double-rotation	1.4661E-5	2.6242E-7	8.1490E-6	4.2156E-6
		Triple-rotation	1.1167E-5	2.9820E-7	4.5037E-6	2.7944E-6
	32	Conventional	2.3007E-10	1.3318E-11	1.2682E-10	7.3241E-11
		Double-rotation	2.2609E-10	2.3192E-11	1.284E-10	6.1322E-11
		Triple-rotation	1.7264E-10	1.0891E-11	1.0442E-10	5.5707E-11
	64	Conventional	1.6653E-16	0	4.9564E-17	5.1286E-17
		Double-rotation	1.1102E-16	0	5.0885E-17	3.6011E-17
		Triple-rotation	8.3267E-17	0	2.9077E-17	2.5174E-17

3.6 The Linear Coordinate System

Basic linear functions can be performed by the CORDIC algorithm as shown in Tab. 3.11. The linear multiplication function (the function number 1) and the linear division function (the function number 2) can be formulated in rotation mode and vectoring mode. Some of the published literatures have described the use of the two basic linear functions. A. Angarita [2] applied the CORDIC algorithm on the circular and linear coordinate systems for broadband communication system. The linear division function was employed for inverse operator $(\frac{1}{X})$ for the received preamble channel. In addition, the linear division function was also utilized to implement a fast inversion and division operators for floating-point computation [120] [81], where these operators are a part of an arithmetic unit of *SPECTRON* streaming processor designed specific for adaptronic applications [143]. The equations of the double-rotation and triple-rotation methods are expressed in Equations (3.28) and (3.29).

The micro-rotation of double-rotation method on the linear coordinate system

$$\begin{aligned} x_{i+1} &= x_i \\ y_{i+1} &= y_i + \delta_i \cdot 2^{-i} \cdot x_i \\ z_{i+1} &= z_i - \delta_i \cdot 2 \cdot 2^{-i-1} \end{aligned} \tag{3.28}$$

The micro-rotation of triple-rotation method on the linear coordinate system

$$\begin{aligned} x_{i+1} &= x_i \\ y_{i+1} &= \delta_i \cdot x_i \cdot (2^{-i-1} + 2^{-i-2}) + y_i \\ z_{i+1} &= z_i - \delta_i \cdot 3 \cdot 2^{-i-2} \end{aligned} \tag{3.29}$$

Rotation mode, $z \longrightarrow 0$

$$\delta_i = \begin{cases} -1 & for\, z_i < 0, \\ 1 & otherwise. \end{cases} \tag{3.30}$$

Vectoring mode, $y \longrightarrow 0$

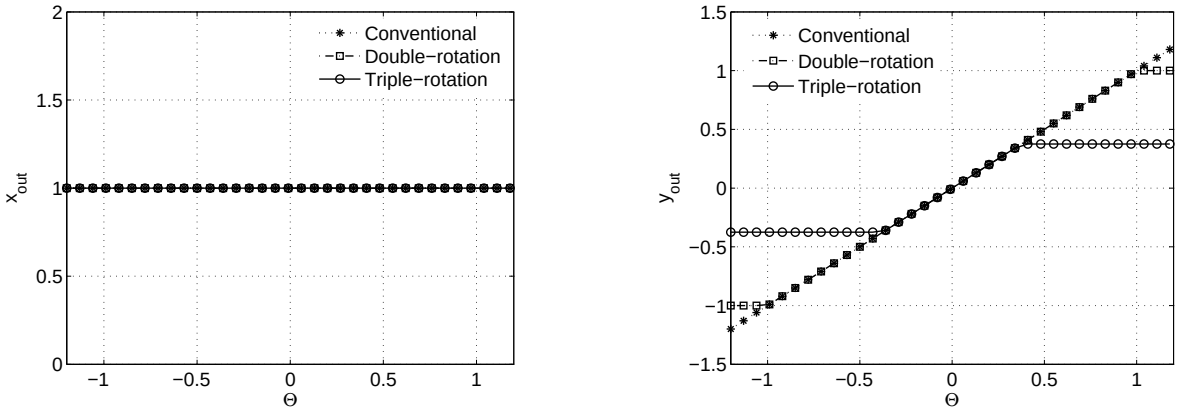
$$\delta_i = \begin{cases} 1 & for\, y_i < 0, \\ -1 & otherwise. \end{cases} \tag{3.31}$$

3.6.1 Convergence

The function numbers 1 and 2 executing in rotation mode and in vectoring mode are modeled and simulated by Matlab/Simulink to consider their convergence ranges, as shown in Figs. 3.15 and 3.16. The x_{in}, y_{in}, z_{in}, which are inputs of the function number 1, are initialized as follows: $x_{in} = cos(\Theta)$ $y_{in} = 0$, and $z_{in} = sin(\Theta)$. The inputs of

Tab. 3.11: The elementary functions with initial parameters for rotation mode and vectoring mode on the linear coordinate system of CORDIC.

No.	Functions	δ_i	Initial parameters		
			x_{in}	y_{in}	z_{in}
1	$x_{out} = x_{in}$ $y_{out} = y_0 + x_0 \cdot z_0$	Sign(z_i)	given value	given value	given value
2	$x_{out} = x_{in}$ $z_{out} = z_0 + \frac{y_0}{x_0}$	Sign(y_i)	given value	given value	given value



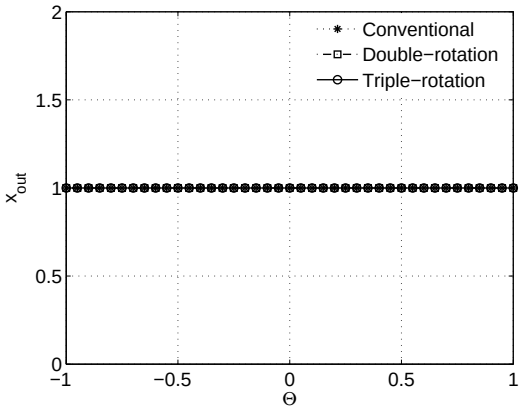
(a) x_{in} (b) $y_{in} + x_{in} \cdot z_{in}$

Fig. 3.15: Convergence ranges of linear function performed by the three CORDIC methods in rotation mode, $z_i \longrightarrow 0$.

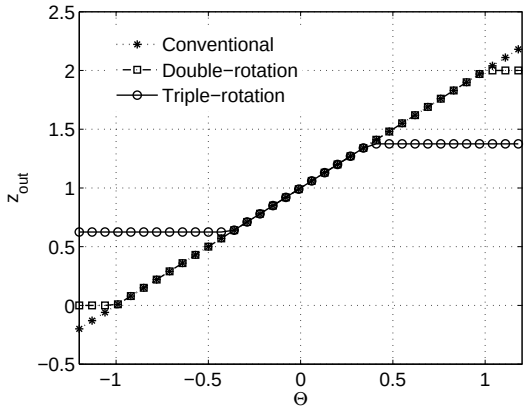
the function number 2, x_{in}, y_{in}, z_{in}, are set as follows: $x_{in} = cos(\Theta)$ $y_{in} = sin(\Theta)$, and $z_{in} = 0$. The simulation results of the two functions show that the convergence ranges Θs of the double-rotation and the triple-rotation methods are $-0.9885 \leq \Theta \leq +0.9885$ and $-0.3747 \leq \Theta \leq +0.3747$, respectively. Figs. 3.17 and 3.18 illustrate the convergence behavior of the three parameters, x_i, y_i, z_i, on the CODIC methods. The triple-rotation method converges faster than the double-rotation method and the double-rotation method converges faster than the convention method.

3.6.2 Accuracy

The computational accuracy of the CORDIC method in rotation and vectoring modes on the linear coordinate system is considered. The *MAPE* is used to measure the computational accuracy when the number of iterations *N* is varied from 8 to 64 as illustrated in Tab. 3.12. The *MAPE* shows that the double-rotation and triple-rotation methods can provide significantly higher computational accuracy than the conventional method with the small number of iterations. Moreover, the computational accuracies of the linear

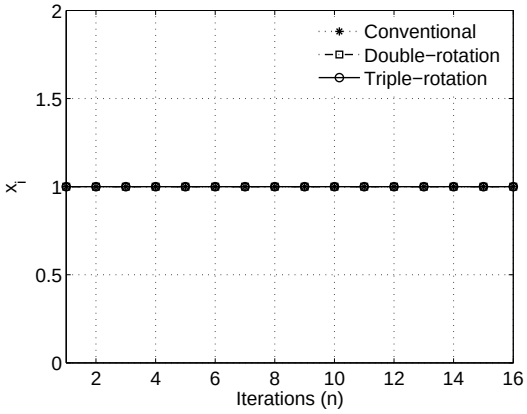


(a) x_{in}

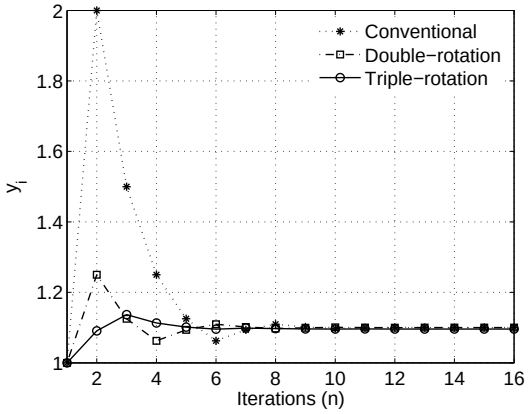


(b) $z_{in} + \frac{y_{in}}{x_{in}}$

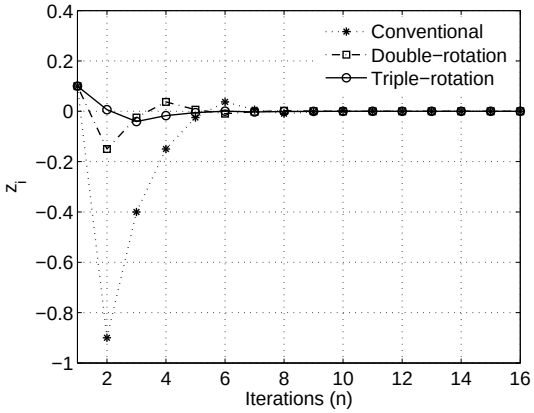
Fig. 3.16: Convergence range of linear multiplication function performed by the conventional and double-rotation CORDIC algorithm in vectoring mode, $y_i \longrightarrow 0$.



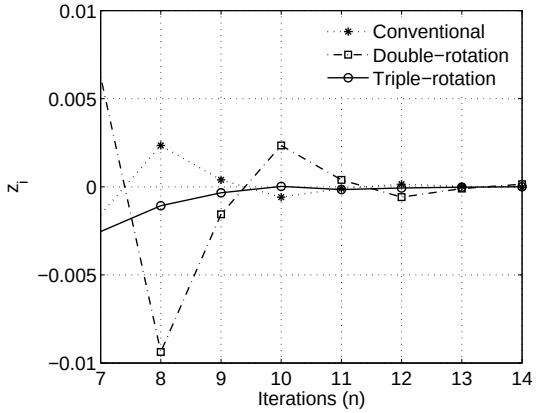
(a) Convergence of x_i



(b) Convergence of y_i

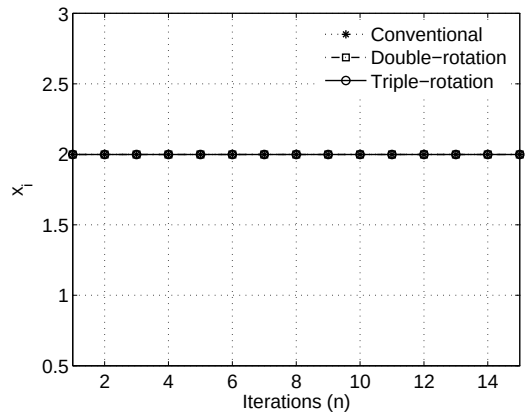


(c) Convergence of z_i

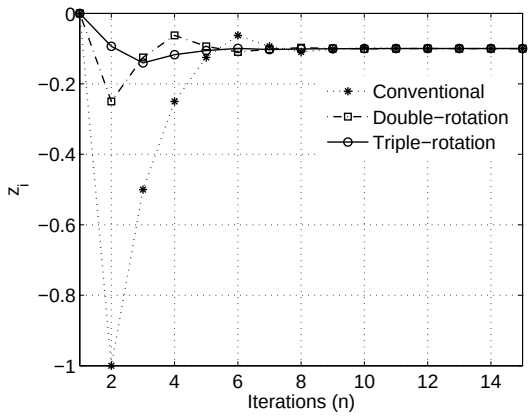


(d) Zoomed convergence of z_i

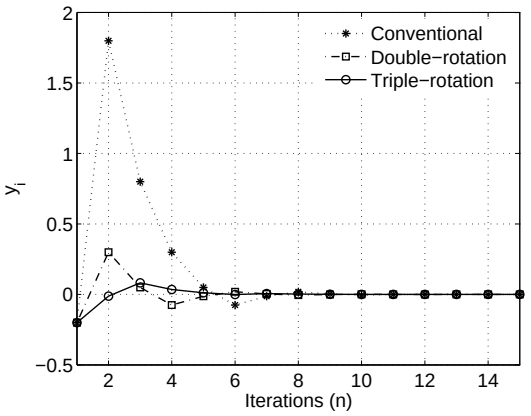
Fig. 3.17: Convergence parameters x_i, y_i, z_i of function $z_{in} + (y_{in} \cdot x_{in})$ performed by the three CORDIC methods, where $x_{in} = 1.0$, $y_{in} = 1.0$, $z_{in} = 0.1$



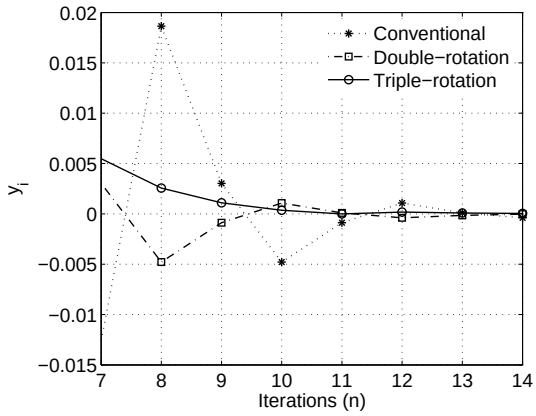
(a) Convergence of x_i



(b) Convergence of z_i



(c) Convergence of y_i



(d) Zoomed convergence of y_i

Fig. 3.18: Convergence parameters x_i, y_i, z_i of function $z_{in} + \frac{y_{in}}{x_{in}}$ performed by the three CORDIC methods, where $x_{in} = 1.999$, $y_{in} = -0.2$, $z_{in} = 0$

Tab. 3.12: The *MAPE* comparisons of y_i and z_i of the conventional, double-rotation and triple-rotation CORDIC methods in the linear coordinate system, where the number of iterations *N* equals to 8, 10, and 16.

CORDIC		*Mean absolute percent error (MAPE)*				
		N=8	*N=10*	*N=16*	*N=32*	*N=64*
Rotation mode						
y_i	Conventional	6.8483%	1.0266%	0.04718%	5.2136E-7%	3.4457E-14%
	Double-rotation	4.1144%	1.5457%	1.4901E-2%	1.0505E-7%	1.8486E-14%
	Triple-rotation	5.8293E-1%	3.4659E-1%	9.1797E-3%	6.4208E-8%	2.4072E-14%
Vectoring mode						
z_i	Conventional	3.3082E-1%	8.2706E-2%	1.2923E-3%	1.9719E-8%	7.1168E-15%
	Double-rotation	1.7091E-1%	4.2729E-2%	6.6764E-4%	1.0187E-8%	7.1168E-15%
	Triple-rotation	5.8530E-2%	1.2635E-2%	2.2863E-4%	3.4886E-9%	1.1452E-14%

multiplication function and the linear division function are evaluated by the four statistical indicators, i.e. maximum absolute error $(Max.|error|)$, minimum absolute error $(Min.|error|)$, average absolute error $(Ave.|error|)$, and standard deviation absolute error $(Std.\,Dev.|error|)$, as shown in Equations (3.16)-(3.19). The tables obviously show that the double-rotation and triple-rotation methods have higher precision than the conventional method.

Tab. 3.13: The computational accuracy analysis of the three CORDIC methods in rotation mode on the linear coordinate system.

	Iteration (N)	CORDIC	Statistical Analysis			
			$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
y_{out}	8	Conventional	7.8125E-3	3.125E-4	3.9405E-3	2.2457E-3
		Double-rotation	3.9063E-3	1.5625E-4	1.9093E-3	1.1437E-3
		Triple-rotation	1.4648E-3	5.8594E-5	7.3695E-4	4.2359E-4
	10	Conventional	1.95310E-3	7.8125E-5	9.7752E-4	5.7493E-4
		Double-rotation	9.76560E-4	3.9063E-5	4.9638E-4	2.7927E-4
		Triple-rotation	9.76560E-4	3.9063E-5	4.9638E-4	2.7927E-4
	16	Conventional	3.0518E-5	1.2207E-6	1.5869E-5	8.7517E-6
		Double-rotation	1.5259E-5	6.1035E-7	7.4582E-6	1.6659E-6
		Triple-rotation	5.7220E-6	7.6294E-8	2.7819E-6	6.7708E-5
	32	Conventional	4.6566E-10	1.8626E-11	2.2942E-10	1.3055E-10
		Double-rotation	2.3283E-10	9.3132E-12	1.1471E-10	5.9722E-11
		Triple-rotation	8.7311E-11	3.4924E-12	4.3585E-11	2.5047E-11
	64	Conventional	5.5511E-17	0	2.9112E-17	2.3385E-17
		Double-rotation	2.7756E-17	0	2.7092E-18	7.7385E-18
		Triple-rotation	2.7756E-17	0	1.3624E-17	1.0809E-17

Tab. 3.14: The computational accuracy analysis of the three CORDIC methods in vectoring mode on the linear coordinate system.

	Iteration (N)	CORDIC	Statistical Analysis			
			$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
z_{out}	8	Conventional	7.8125E-3	1.5625E-3	4.0625E-3	2.6146E-3
		Double-rotation	3.9063E-3	7.8125E-4	2.0313E-3	1.3073E-3
		Triple-rotation	1.0742E-3	2.9297E-4	6.8359E-4	3.0882E-4
	10	Conventional	1.9531E-3	3.9063E-4	1.0156E-3	6.5364E-4
		Double-rotation	9.7656E-4	1.9531E-4	5.0781E-4	3.2682E-4
		Triple-rotation	3.1738E-4	2.4414E-5	1.5137E-4	1.1759E-4
	16	Conventional	3.0518E-5	6.1035E-6	1.5869E-5	1.0213E-5
		Double-rotation	1.5259E-5	3.0518E-6	7.9346E-6	5.1066E-6
		Triple-rotation	4.1962E-6	1.1444E-6	2.6703E-006	1.2063E-6
	32	Conventional	4.6566e-10	9.3132E-11	2.4214E-10	1.5584E-10
		Double-rotation	2.3283e-10	4.6566E-11	1.2107E-10	7.792E-11
		Triple-rotation	6.4028e-11	1.7462E-11	4.0745E-11	1.8407E-11
	64	Conventional	2.2204E-16	0	8.8818E-17	1.2162E-16
		Double-rotation	2.2204E-16	0	8.8818E-17	1.2162E-16
		Triple-rotation	2.2204E-16	0	1.3323E-16	1.2162E-16

3.7 Unified CORDIC

In this section, the unified micro-rotations of the double-rotation and triple-rotation CORDIC methods are introduced. They are described in the circular, hyperbolic, and linear coordinate systems in a unified manner by defining the parameter *m* as:

- *m=1*: for the circular coordinate system
- *m=0*: for the linear coordinate system
- *m=-1*: for the hyperbolic coordinate system

The unified micro-rotation of the double-rotation method

$$\begin{aligned} x_{i+1} &= x_i - m \cdot (\delta_i \cdot 2^{-i} \cdot y_i + 2^{-2i-2} \cdot x_i) \\ y_{i+1} &= y_i + \delta_i \cdot 2^{-i} \cdot x_i - m \cdot 2^{-2i-2} \cdot y_i \\ z_{i+1} &= \begin{cases} z_i - \delta_i \cdot 2 \cdot tan^{-1}(2^{-i-1}) & if\ m = 1 \\ z_i - \delta_i \cdot 2 \cdot tanh^{-1}(2^{-i-1}) & if\ m = -1 \\ z_i - \delta_i \cdot 2^{-i} & if\, m = 0 \end{cases} \end{aligned} \tag{3.32}$$

The unified micro-rotation of the triple-rotation method

$$\begin{aligned} x_{i+1} &= x_i \cdot (1 - m \cdot (2^{-2i-3} + 2^{-2i-4})) - m \cdot \delta_i (2^{-i-1} + 2^{-i-2} - m \cdot 2^{-3i-6}) \cdot y_i \\ y_{i+1} &= \delta_i \cdot x_i \cdot (2^{-i-1} + 2^{-i-2} - m \cdot 2^{-3i-6}) + (1 - m \cdot (2^{-2i-3} + 2^{-2i-4})) \cdot y_i \\ z_{i+1} &= \begin{cases} z_i - \delta_i \cdot (2 \cdot tan^{-1}(2^{-i-2}) + tan^{-1}(2^{-i-2})) & if\ m = 1 \\ z_i - \delta_i \cdot (2 \cdot tanh^{-1}(2^{-i-2}) + tanh^{-1}(2^{-i-2})) & if\ m = -1 \\ z_i - \delta_i \cdot (2^{-i-1} + 2^{-i-2}) & if\ m = 0 \end{cases} \end{aligned} \tag{3.33}$$

With the non-redundant method, the scaling factors of the double-rotation and triple-rotation methods are $K_{dr} = \prod_{i=1}^{n} \frac{1}{(1+2^{-2i-2})}$ and $K_{tr} = \prod_{i=2}^{n+1} \frac{1}{(1+2^{-2i-4})^{\frac{3}{2}}}$ which are approximately 0.9219 and 0.992, respectively.

3.8 Extension Functions

3.8.1 Natural Logarithm

The generations of elementary functions such as *log()* and *antilog()* are used in many scientific applications such as digital signal processing, 3-D computer graphics, scientific computing, artificial neural networks, logarithmic number system (*LNS*), and other multimedia applications [7]. For instance, the functions were utilized for computational probability of bit error to analyse performance of the anti-jam communication system [116] [130].

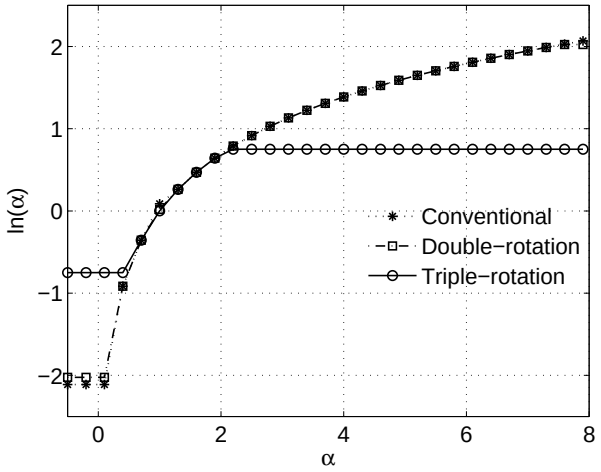


Fig. 3.19: The simulation result of the natural logarithmic function based on the CORDIC methods in the vectoring mode on the hyperbolic coordinate system.

The natural logarithmic function is also an importance part of a computational unit in image processing applications. The function was employed to perform the bilateral neighborhood filter for biomedical image processing [41], the image enhancement and segmentation for remote sensing [9], the *LIP* tool for in-camera image processing [23], etc. The method widely used to perform the natural logarithm is the *LUT-based* method, which is an approximation. Some of the previous works involving *LUT-based* methods, *LUTs* combined with polynomial approximation [115] [92], symmetric bipartite table-based approximations [106], etc. However, the *LUT-based* method requires a large memory to contain information for computation. To avoid this problem, this section introduce the natural logarithmic function performed by the CORDIC methods. Algorithm 9 depicts the natural logarithmic algorithm based on the function of CORDIC in vectoring mode on the hyperbolic coordinate system whereas α is an input of the algorithm.

Algorithm 9 $y = Ln(\alpha)$

1: $x_{in} = \alpha + 1$;
2: $y_{in} = \alpha - 1$;
3: $z_{in} = 0$;
4: $z_{out} = z_{in} + tanh^{-1}(\frac{y_{in}}{x_{in}})$;
{CORDIC algorithm in vectoring mode on the hyperbolic coordinate system}
5: $y = 2 \cdot z_{out}$;
6: return y;

The convergence ranges of the conventional, double-rotation, and triple-rotation methods are in [0.1211,8.2562], [0.132,7.565], and [0.4721,2.1182] respectively as shown in Fig. 3.19. The functions can work efficiently with the CORDIC method in conventional and double-rotation due to the convergence range results. Their computational accuracies are evaluated by the *MAPE* factor and the four statistical analysis indicators reported in Tabs. 3.15

Tab. 3.15: The *MAPE* comparisons of the natural logarithmic function performed by the conventional, double-rotation and triple-rotation methods with the number of iterations *N* varied from 8 to 64.

CORDIC		*Mean absolute percent error (MAPE)*				
		N=8	*N=10*	*N=16*	*N=32*	*N=64*
$Ln(\alpha)$	Conventional	1.1864%	3.2707E-1%	7.9463E-3%	2.658E-2%	2.5657E-3%
	Double-rotation	1.1982%	2.9438E-1%	5.1638E-3%	1.0844E-4%	1.0837 E-4%
	Triple-rotation	9.2564E-1%	1.8985E-1%	3.2671E-3%	5.3340E-8%	1.9278E-14%

and 3.16, where the convergence range of the triple-rotation method is used in simulation. Although the accuracy of the triple-rotation method is better than the conventional and double-rotation methods in *MAPE* and the four statistical indicators, its convergence range is very small.

3.8.2 Square Root

This function is also another function that can be built by the CORDIC method in vectoring mode on the hyperbolic coordinate system. Similar to the natural logarithmic function, the *LUT-based* method is applied to built this function, where the large memory storage is required to keep the information for computation. There is another method to create square root function such as the *Generalized Svoboda and Tung* (*GST*) division method [62]. Some of the scientific applications use, the square root *Bryson-Frazier Smoothing* algorithm [96]. For example, the square root function was used to solve the nonlinear attitude error measurement equations on geomagnetism, *GPS*, *SINs* navigation systems [51], the fast square-root detection algorithm [135]. The square root function based on the CORDIC method was introduced by Min Ye et al. [133]. The function is used to extract parameters of digital pre-distortion for power amplifiers. In this section, the CORDIC-based method is figured out based on the conventional, double-rotation, and triple-rotation methods. The CORDIC-based square root function is illustrated in Algorithm 10.

Algorithm 10 $y = \sqrt{\alpha}$

1: $x_{in} = \alpha + 0.25$;
2: $y_{in} = \alpha - 0.25$;
3: $x_{out} = \sqrt{x_{in}^2 - y_{in}^2}$;
{CORDIC algorithm in vectoring mode on the hyperbolic coordinate system}
4: $y = x_{out}$;
5: return y;

Fig. 3.20 shows the simulation results of the CORDIC-based square root function on Matlab/Simulink. Like the natural logarithmic function, the convergence range of the

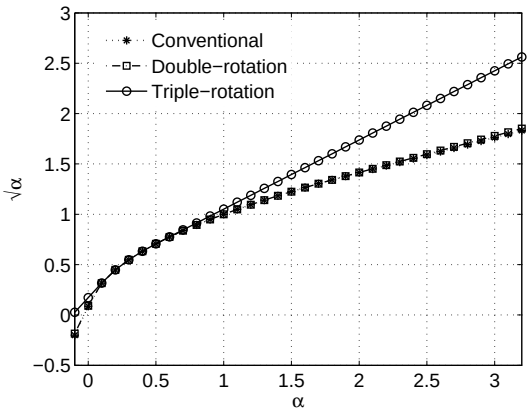
Tab. 3.16: The statistical analysis of computational accuracy of the natural logarithmic function.

	Iteration (N)	CORDIC	Statistical Analysis			
			$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
$Ln(\alpha)$	8	Conventional	8.9405E-3	3.5196E-6	3.9273E-3	2.3300E-3
		Double-rotation	7.8637E-3	8.5728E-5	3.9081E-3	2.2538E-3
		Triple-rotation	5.8149E-3	1.0345E-5	2.9487E-3	1.7197E-3
	10	Conventional	3.0811E-3	1.8507E-5	1.0535E-3	6.0042E-4
		Double-rotation	2.0043E-3	2.0237E-5	9.6088E-4	5.5240E-4
		Triple-rotation	1.4545E-3	3.5376E-6	7.5850E-4	4.3871E-4
	16	Conventional	1.1585E-3	4.9451E-7	2.4967E-5	9.1134E-5
		Double-rotation	8.1678E-5	1.7282E-7	1.5741E-5	1.0376E-5
		Triple-rotation	2.2622E-5	1.5712E-7	1.0616E-5	6.4114E-6
	32	Conventional	0.001128	4.7332E-12	7.8754E-6	8.9513E-5
		Double-rotation	5.1161E-5	3.5842E-12	3.18E-7	4.0321E-6
		Triple-rotation	3.4895E-10	1.3366E-12	1.7805E-10	9.1915E-11
	64	Conventional	0.001128	0	7.8752E-6	8.9513E-5
		Double-rotation	5.1161E-5	0	3.1777E-7	4.032E-6
		Triple-rotation	3.3307E-16	0	6.9217E-17	7.2178E-17

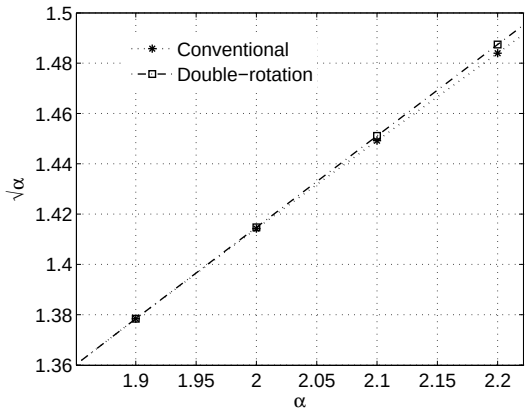
triple-rotation method is smaller than the convergence range of the conventional and double-rotation methods. The computational accuracy is considered by the *MAPE* and the four statistical indicators as shown in Tab. 3.18. Although the triple-rotation method provides higher accuracy than the conventional and double-rotation methods, its convergence range becomes inefficiency. Thus the conventional and double-rotation methods are suitable in practice, whereas the double-rotation method is only applied for high computational accuracy.

Tab. 3.17: The *MAPE* comparisons of the square-root function performed by the conventional, double-rotation and triple-rotation CORDIC with the iteration steps *N* varied from 8 to 64 and convergence range from 0.1 to 0.5.

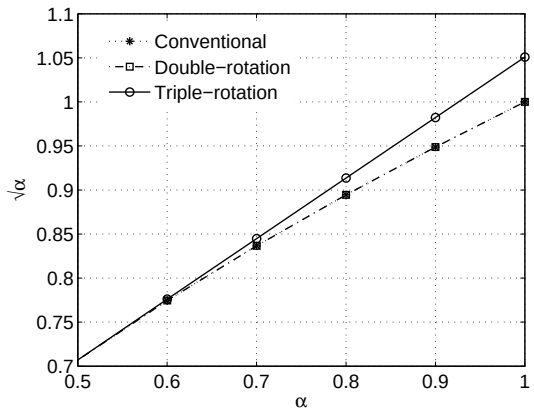
CORDIC		*Mean absolute percent error (MAPE)*				
		n=8	*N=10*	*N=16*	*N=32*	*N=64*
$\sqrt{\alpha}$	Conventional	4.8701E-3%	3.8562E-3%	3.6089E-3%	3.6053E-3%	3.6053E-3%
	Double-rotation	5.6474E-4%	1.5769E-4%	1.1179E-4%	1.1143E-4%	1.1143E-4%
	Triple-rotation	9.6425E-5%	8.8287E-5%	5.2716E-5%	7.25440E-5%	9.0528E-5%



(a) convergence range on the three CORDIC algorithm



(b) the broke of double-rotation from conventional



(c) the broke of triple-rotation from conventional and double-rotation

Fig. 3.20: The convergences of the square root function based on the conventional, double-rotation, and triple-rotation methods

Tab. 3.18: The computational accuracy analysis of the square root function.

	Iteration (N)	CORDIC	Statistical Analysis			
			$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.Dev.\|error\|$
$\sqrt{\alpha}$	8	Conventional	5.5351E-4	1.3214E-11	2.9934E-5	9.3537E-5
		Double-rotation	4.4429E-5	4.2641E-7	2.4859E-6	4.1224E-6
		Triple-rotation	1.1568E-3	7.4691E-7	3.5948E-5	1.3121E-4
	10	Conventional	4.8671E-4	1.0009E-11	2.3891E-5	8.0145E-5
		Double-rotation	2.6596E-5	4.3396E-8	4.4852E-7	2.2364E-6
		Triple-rotation	1.0979E-3	3.9090E-6	3.4721E-5	1.2287E-4
	16	Conventional	4.6573E-4	5.5511E-16	2.2300E-5	7.5948E-5
		Double-rotation	2.1836E-5	1.7065E-8	2.5739E-7	1.7703E-6
		Triple-rotation	1.0789E-3	4.4772E-6	3.4165E-5	1.2024E-4
	32	Conventional	4.6540E-4	0	2.2277E-5	7.5882E-5
		Double-rotation	2.1765E-5	1.7048E-8	2.5624E-7	1.7633E-6
		Triple-rotation	1.0786E-3	4.4434E-6	3.4156E-5	1.2019E-4
	64	Conventional	4.6540E-4	0	2.2277E-5	7.5882E-5
		Double-rotation	2.1765E-5	1.7048E-8	2.5624E-7	1.7633E-6
		Triple-rotation	1.0786E-3	4.4434E-6	3.4156E-5	1.2019E-4

3.9 Problem of Convergence Range on Elementary Functions

The CORDIC algorithm is a very powerful tool in scientific applications such as robot control [44] [64] [119], engineering graphics [31] [112] and digital signal processing [9] [41]. However, a major shortcoming of CORDIC algorithm is the magnitude restrictions as shown on the previous sections, where there is a limitation of the convergence range available for the CORDICs' input parameters. In this section, extension of the CORDICs' convergence range is discussed in order to improve performance. There are two different approaches which are applied to solve the limitation of the convergence range of the CORDIC algorithm, i.e. pre/post processing with mathematical identities and expansion of the convergence range. Both methods require a significant amount of overhead processing time and extra hardware for *VLSI* implementation.

3.9.1 Pre/post Processing with Mathematical Identities Method

This method was introduced in few literatures [126] [47] [114], where mathematic identities are applied to adapt or adjust input variables x_{in}, y_{in}, z_{in}. This method requires pre-processing for input variable's adaption and post-processing for compensation on output variables x_{out}, y_{out}, z_{out}. This technique works very well for transcendental func-

tions performed by CORDIC algorithm in rotation mode such as sine, cosine, etc. In this case study, the convergence range extension based on the pre/post processing with mathematical identities method is examined. It is applied for the sine and cosine function generators based on triple-rotation method considered in Algorithm 11.

Algorithm 11 [$cos(\phi), sin(\phi)$]=Sin-Cos(ϕ, I)

1: y_2=0, z_2=0, x_2=$\frac{1}{K_{tr}}$, $\beta = 0.3747$ {Initialization}
2: [ϕ_p, neg] = *PhaseDetect*(ϕ)
3: [ϕ_q, x_s, y_s] = *QuadrantDetect*(ϕ_p)
4: [ϕ_{cordic}] = *PreCal*(ϕ_q, β)
5: Z_2=ϕ_{cordic}
6: **for** $i = 2$ to I **do**
7: **if** $Z_i < 0$ **then**
8: δ_i=-1
9: **else**
10: δ_i=1
11: **end if**
12: $x_{i+1} = x_i(1 - 2^{-2i-3} - 2^{-2i-4}) - \delta_i(2^{-i-1} + 2^{-i-2} - 2^{-3i-6})y_i$
13: $y_{i+1} = \delta_i x_i(2^{-i-1} + 2^{-i-2} - 2^{-3i-6}) + (1 + 2^{-2i-3} - 2^{-2i-4})y_i$
14: $z_{i+1} = z_i - \delta_i((2tan^{-1}(2^{-i-2}) + tan^{-1}(2^{-i-2}))$
15: **end for**
16: [$cos(\phi), sin(\phi)$] = *PostCal*($x_I, y_I, x_s, y_s, neg, \phi, \beta$)
17: return $cos(\phi), sin(\phi)$

$x_s = -1$ if $\frac{\pi}{2} < \phi_p \leq \frac{3\pi}{2}$; otherwise 1. $y_s = -1$ if $\pi < \phi_p \leq 2\pi$; otherwise 1. The *PreCal* function adapts ϕ_q to the triple-rotation method range $\beta = 0.3747$ as illustrated in Equation (3.34).

$$\phi_{cordic} = \begin{cases} \phi_q & , 0 \leq \phi_q \leq \beta \\ \phi_q - \beta & , \beta < \phi_p \leq 2\beta \\ \phi_q - 2\beta & , 2\beta < \phi_p \leq 3\beta \\ \phi_q - 3\beta & , 3\beta < \phi_p \leq 4\beta \\ \phi_q - 4\beta & , 4\beta < \phi_p \leq 5\beta \end{cases} \tag{3.34}$$

The *PostCal* function performs final calculation results of $Cos(\phi)$ and $Sin(\phi)$ corresponding to the CORDIC results, the phase and quadrant values, the angle ϕ and the valid angle β of the pre-processing steps. In *VLSI* practice, $Sin(\beta)$ and $Cos(\beta)$ are performed as constant in an off-line computation.

Since this method requires the pre-processing and the post-processing, the computational accuracy corresponds to the constant values used for computation with these

Algorithm 12 $[x_{new}, y_{new}]$=PostCal$(x, y, x_s, y_s, neg, \phi, \beta)$

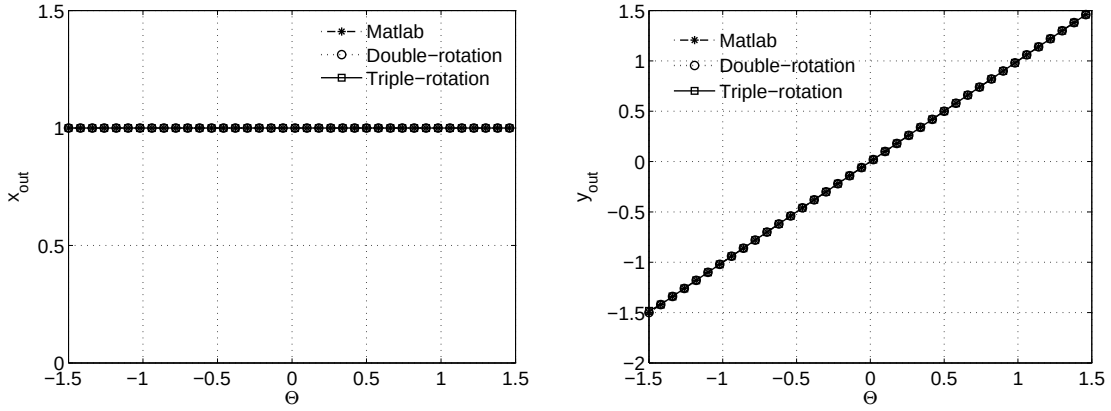
if $(\phi \geq 0)$ and $(\phi \leq \beta)$ **then**
    x_{new}=x, y_{new}=y
else if $(\phi \geq \beta)$ & $(\phi \leq 2\beta)$ **then**
    x_{new}=$x \cdot cos(\beta) - y \cdot sin(\beta)$, y_{new}=$x \cdot sin(\beta) + y \cdot cos(\beta)$
else if $(\phi \geq 2\beta)$ & $(\phi \leq 3\beta)$ **then**
    x_{new}=$x \cdot cos(2\beta) - y \cdot sin(2\beta)$, y_{new}=$x \cdot sin(2\beta) + y \cdot cos(2\beta)$
else if $(\phi \geq 3\beta)$ & $(\phi \leq 4\beta)$ **then**
    x_{new}=$x \cdot cos(3\beta) - y \cdot sin(3\beta)$, y_{new}=$x \cdot sin(3\beta) + y \cdot cos(3\beta)$
else if $(\phi \geq 4\beta)$ **then**
    x_{new}=$x \cdot cos(4\beta) - y \cdot sin(4\beta)$, y_{new}=$x \cdot sin(4\beta) + y \cdot cos(4\beta)$
else
    x_{new}=x, y_{new}=y
end if
$x_{new} = x_{new} \cdot x_s$
$y_{new} = y_{new} \cdot y_s \cdot neg$
return x_{new}, y_{new}

processes. Thus, the precision of the constant values has to be considered case-by-case, where this area is still available for future research.

3.9.2 Sequential Index Extension Method

In this process, a set of iteration indexes i is considered, where the CORDIC's variables x_i, y_i, and z_i will be updated based on the iteration indexes i. With different set of the iteration indexes, the CORDIC results in different convergence ranges. The CORDIC based on the sequential index extension method was published in few articles [50] [28] [83], where a variable, which is either y or z, will be driven toward to zero in either vectoring mode or rotation mode. Xiaobo Hu [50] introduced a series of linear indexes for the CORDIC on the liner coordinate system. The set of the linear indexes is expanded by $i = -M, -M+1, \cdots, N$, where M and N are two integers involving in the convergence range extension.

Similarly to the convergence range of the CORDIC in linear coordinate system, this method can be applied for the circular and hyperbolic coordinate systems. For example, to perform the *Rectangular-to-Polar* function (CORDIC in vectoring mode on the circular coordinate system) by the double-rotation and triple-rotation methods with 16 iterations, their convergence range can be extended from -1.5 to 1.5 by determining the sequential indexes, $[-1, 0, \cdots, 14]$ and $[-2, 0, \cdots, 13]$. Their magnitude results are subtracted by a constant value, i.e. 1.5 for the double-rotation method and 0.5666 for the triple-rotation method. The simulation result is illustrated in Fig. 3.21. However, the convergence range extensions and the computational accuracy evaluations based on the proposed double-



(a) $x_{out} = \sqrt{x_{in}^2 + y_{in}^2}$ (b) $y_{out} = z_{in} + tan^{-1}\frac{y_{in}}{x_{in}}$

Fig. 3.21: The convergence range extensions of the double-rotation and triple-rotation CORDIC methods based on the sequential index extension method.

rotation and triple-rotation methods have to be further investigated and examined case-by-case.

3.10 Summary

This chapter discusses the CORDIC algorithms and elementary functions based on non-redundant mechanism, where rotation direction δ is in a set of 1 and -1. The two CORDIC methods are proposed, i.e. the double-rotation and triple-rotation. The non-redundant technique is applied in order to compute constant scaling factors. The performance and time efficiency of the proposed methods are investigated and evaluated based on Matlab/Simulink ideal results by considering the convergence range, the computational latency and the computational accuracies. Moreover, the unified micro-rotation and the extension functions based on the double-rotation and triple-rotation methods are designed and described. Finally, the convergence range extension is discussed and proposed for future research. The contributions on this chapter are summarized as follows.

- *Convergence*: Since the double-rotation and triple-rotation methods are the acceleration of rotation angle by duplication 2θ and triplication 3θ, the two methods respond to the parameters' convergence range rather smaller than the conventional method. The simulation results show that the convergence ranges of the proposed methods are in between [-0.9885,+0.9885] and [-0.3747,+0.3747] for the double-rotation and triple-rotation methods, respectively. The convergence ranges are used to evaluated the initial input variables x_{in} and y_{in} in rotation mode ($z_{in} \longrightarrow 0$) and the initial input variables x_{in} and z_{in} in vectoring mode ($y_{in} \longrightarrow 0$) to grantee that the CORDIC methods can provide correct computational results. However, the proposed methods provide faster convergence than the conventional method in the parameters x,

y, and z.

- *Computational latency and accuracy*: Due to the faster parameter's convergence, the double-rotation and triple-rotation methods provide higher computational accuracy than the conventional method at the same number of iterations. In turn, the methods can perform computational results with the smaller number of iterations at the same expected error. For analysis, *Mean-Absolute-Percentage-Error (MAPE)* and the statistical analysis factors, i.e. maximum absolute error $(Max.|error|)$, minimum absolute error $(Min.|error|)$, average absolute error $(Ave.|error|)$, and standard deviation absolute error $(Std.\,Dev.|error|)$ are applied for the investigation, as they are convenient in practice due to the nonlinear equation problem of the CORDIC's mathematical assumption.

- *The unified micro-rotation and the extension functions*: The unified micro-rotation of the double-rotation and triple-rotation methods are proposed, where the parameter *m* is introduced for configuration of the CORDIC's coordinate systems. According to the advantage of the proposed method dealing with the high computational accuracy and the fast convergence in the parameters *x, y, z*, they will be applied to design and implement the high and fast computational accuracy CORDIC core as described in chapter 4. The algorithms for the two extension functions, i.e. natural logarithm and square root, based on the suggested CORDIC methods are proposed and investigated. The computational results of the two algorithms are compared with Matlab/Simulink ideal results, where the double-rotation provides better performance and efficiency than the triple-rotation.

- *The convergence range extension*: Although the double-rotation and triple-rotation methods provide the advantage in the computational accuracy and low latency compared to the conventional method, they lack in convergence range of the parameters *x, y, z*. To alleviate the convergence range problem, the two possible methods, i.e. the pre/post processing with the mathematical identity method and the sequential index extension method are discussed. The sin-cosine algorithm based on the proposed triple-rotation CORDIC method is first introduced and exampled, where its convergence range is extended by the pre/post processing with the mathematical identity method. Meanwhile, the sequential index extension method is also verified with the *Rectangular-to-Polar* function, which is CORDIC's elementary functions in vectoring mode on the circular coordinate system. The function is introduced on both the double-rotation and the triple-rotation methods with the two sequential indexes $[-1, 0, \cdots, 14]$, $[-2, 0, \cdots, 13]$; afterwards the magnitude results are subtracted by constant values, 1.5 for the double-rotation method and 0.5666 for the triple-rotation method. However, the convergence range extensions are independent on each elementary function, where the computational accuracy has to be carefully considered. This research area provides a great opportunity for further investigation.

Chapter 4

Design and Architecture for *VLSI* implementation of an Arithmetic Unit

Contents

This chapter discusses the design and architecture of a floating-point arithmetic unit which can perform the basic mathematic functions frequently used in science and engineering. The unit consists of five arithmetic modules, i.e. floating-point adder, floating-point multiplier, floating-point product-of-sum, floating-point sum-of-product and CORDIC. Since the design and architecture of the first four modules have already been explained in chapter2, this chapter will mostly focus on the last module. The CORDIC module will be implemented based on the proposed CORDIC methods, double-rotation and triple-rotation, as described in chapter 3. It will then be analysed and synthesised based on the pipeline mechanism to achieve several degrees of performance, and performed in the fixed-point format due to the limited available range (convergence range) of the CORDIC. Afterwards, the five arithmetic modules have to be synchronized together, but they have a different data format. Thus, two data converting algorithms employed to convert data from floating-point to fixed-point and from fixed-point to floating-point are introduced in order for each module to work together efficiently. Finally, the floating-point arithmetic unit will be used on both an accelerator and a reconfigurable steaming processor for case study, where the main purposes of the accelerator/processor are to accelerate computation of any main processors and to process streaming data.

4.1 State-of-Art

In advanced scientific applications, very complex algorithms or formulas can be untangled by modern mathematics; they can solve or explain some intricate problems easier than classical ones. With modern mathematics, undefined scientific phenomena in the former can be explained or modelled. Afterwards, the model will be programmed by software in libraries, and executed by either low or high performance machines depending on the required degree of latency. In some real-time applications, such as image processing or computer graphic applications, the execution of their programs with the libraries cannot meet time constraints, leading to a system's failure. In addition, computational accuracy becomes another significant factor especially in aerospace or military applications, where they need precise computational results as much as possible. To achieve computational time and accuracy, hardware coprocessors/processors with high computational accuracy are thus intensively studied by computer scientists. Several literatures regarding the design, architecture, and implementation of the coprocessors/processors have been considered, and the summaries of the published articles related to this work will be described in the following sections.

4.1.1 Design and Implementation of Floating-Point Accelerator and Processor

In 1983, C. Huntsman et al. [52] introduced the floating-point co-processor MC68881 to accelerate mathematical computation of Motorola's microprocessor M68000 family. The co-processor was implemented in an economic architecture conforming to the floating-point IEEE standard, and can support square root, trigonometric, and transcendental functionalities. G. Wolrich et al. [129] proposed the single-chip floating-point co-processor fabricated in the 3-um NMOS technology to support addition/subtraction, multiplication and square root functions. C. Rowen [101] introduced the floating-point accelerator chip R3010, based on the R3000 RISC processor, to the *MIPS* computational system. The R3010 accelerator achieved high-speed arithmetic computation with low decoding instruction overhead and with a high performance compiler, where unnecessary computational processes and memory traffics were eliminated. The IBM company [80] proposed a floating-point unit (*FPU*) to IBM RISC System/6000* (RS/6000), where the performance and efficiency of the computational unit were improved by modifying its floating-point multiply-add-fused (*MAF*) component. The math accelerator WE32106 unit was proposed by P. M. Maurer [73] for the objective of design and verification. The floating-point co-processor TMS390C602A was created by M. Darley et al. [26] to cooperate computation with the Texas Instrument's micro-processor TMS390C601.

From the reviewed literature, the design and architecture for the *VLSI* implementation of floating-point co-processors have received tremendous attention. The co-processors are not only improving the computational performance and efficiency for a main processor, but also reducing redesign cost and time-to-market where a floating-point arithmetic unit in hardware is not embedded.

4.1.2 Accelerator and Processor based on CORDIC

The CORDIC algorithm can perform elementary functions by a shift-added method which is very easy for engineers to implement arithmetic units in hardware [125]. Several pieces of literature considered the design and implementation of arithmetic units, co-processors and processors based on the CORDIC. The floating-point co-processor, named *Gmicro*, was proposed by S. Kawasaki [58] to support the *Gmicro/200* and the *Gmicro/300* in the *TRON* architecture. The *Gmicro* consisted of basic floating-point operators and special floating-point operators performed by the CORDIC. The *TRON* architecture provided the information infrastructure for various layers of machine society. Important scientific functions were created by the arithmetic unit to fulfil the computational demands of several scientific and engineering areas such as civil, industrial, chemical, control, etc. The vector/matrix instruction floating-point co-processor with CORDIC was proposed by T. Nakayama [84]. He designed the co-processor with the pipeline architecture whose arithmetic unit comprised addition, multiplication, square-root, division, and el-

ementary functions. The performance with 6.7 *MFLOPS* at 20 MHz of the co-processor can be achieved by the use of parallel execution. K. Sarrigeorgidi [103] designed and implemented the ultra low power CORDIC processor for advanced adaptive wireless communication algorithms. A key characteristic of the algorithms was the advanced matrix computations, where some of the algebraic matrix problems, i.e. *Householder transformation*, *QR factorization*, and singular value decomposition, were solved by the CORDIC rotation. J. R. Cavallaro et al. [17] introduced the specific CORDIC processor array to compute matrix factorization problems in real-time signal processing applications. The CORDIC processor for fast fourier transform (*FFT*) computation was proposed by Sarmiento et al. [102], where the processor used the gallium arsenide technology for implementation. The *FFT* is a popular mathematic function to solve problems in many digital signal processing applications, such as radar, sonar, spread-spectrum communication, image processing, 3D graphic [94], etc. Therefore, the *FFT* CORDIC processor is very useful.

The reviewed literatures are just examples of the CORDIC research areas, where they try to optimise and customize the design and architecture of the CORDIC module for specific applications. In this chapter, not only are the two general considerations taken into account but also reconfigurability will be included where elementary functions can be easily performed by readily changing the CORDIC's input parameters. Moreover, the design and architecture of the floating-point arithmetic unit for VLSI implementation are proposed, investigated and compared. The unit consists of the adder, the multiplier, the product-of-sum, the sum-of-product and the CORDIC modules. The remainder of this chapter deals with

1) The design and architecture of unified micro-rotation of the double-rotation and triple-rotation methods in fixed-point representation.

2) A high accuracy CORDIC algorithm, performance investigation, comparison of the proposed CORDIC methods.

3) Two conversion algorithms which are used to convert data between floating-point units and a fixed-point unit and vice versa.

4) The design and architecture of an arithmetic accelerator and a reconfigurable streaming processor.

5) The architectural comparisons of CORDIC and the floating-point accelerator and the streaming floating-point processor with the published literature.

4.2 Unified Micro-Rotation Architecture of CORDIC

The unified micro-rotations of the double-rotation and triple-rotation methods are designed and discussed in this section. The computational results of the CORDIC normally

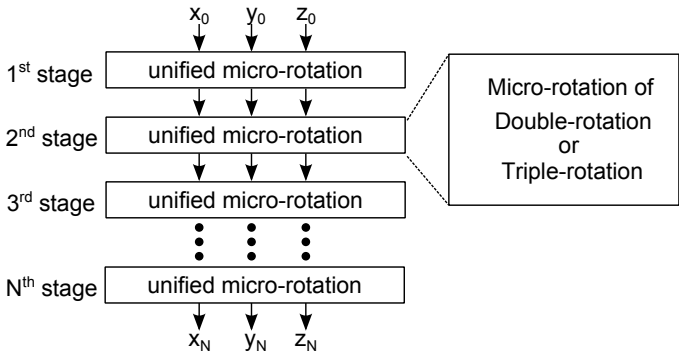


Fig. 4.1: The CORDIC computation in the pipeline architecture.

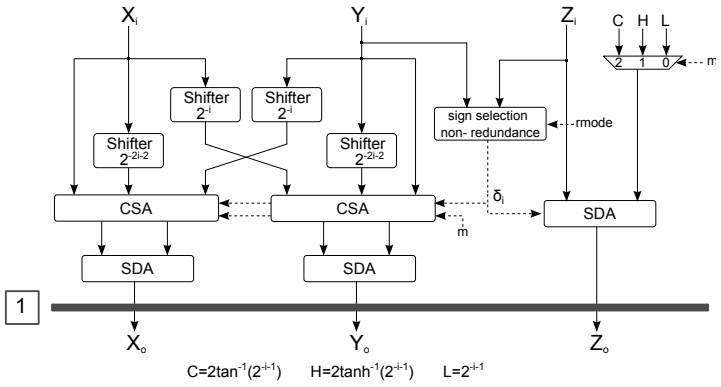
come from iteratively computational rotation of the unified micro-rotation. However, pipeline architecture is applied to achieve the highest throughput. Due to the limited convergence range in parameters x, y, and z of the proposed CORDIC methods which are at [-0.9885,+0.9885] and [-0.3747,+0.3747] for the double-rotation and triple-rotation, the design and architecture of the CORDIC's unified micro-rotation are considered in fixed-point format. Fig. 4.1 shows the diagram of the unified micro-rotation of the double-rotation and triple-rotation methods in the N-stage pipeline architecture.

4.2.1 Design and Architecture

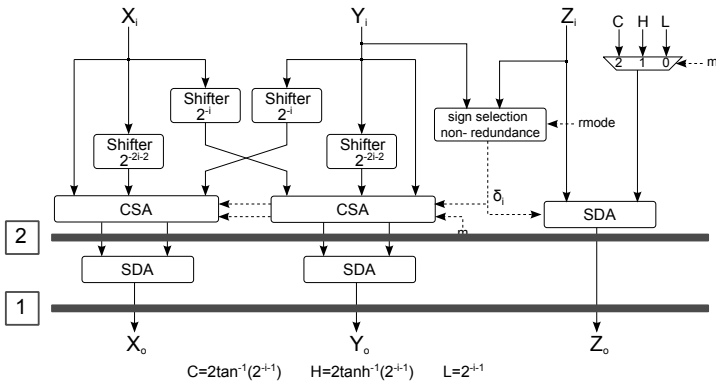
The unified micro-rotation architecture based on the double-rotation CORDIC method in Equation (3.32) is shown in Fig. 4.2. Maximum 3-stage pipeline architectures are proposed. The architecture consists of five main components, i.e. two carry-save-adders (*CSAs*), three sign-digit-adders (*SDAs*), four shifters, one 3-input multiplexer, and a sign selection non-redundant module. The shifter will shift bit of an input binary string from right to left N times whereas N is an integer number which is less than zero. The shift bit from left to right can be done when N is an integer number greater than zero. The rotation direction δ_i will result in either +1 or -1 depending on the parameter *rmode* and either y_i or z_i. The hardware complexity ($A_{double-rotation}$) of the unified micro-rotation of the double-rotation CORDIC method is expressed in Equation (4.1).

Similarly to the unified micro-rotation of the double-rotation method, the pipeline architecture of the unified micro-rotation of the triple-rotation method in Equation (3.12) can be illustrated in Fig. 4.3. The architecture consists of four *CSAs*, seven *SDAs*, ten shifters, one 3-input multiplexer, and a sign selection non-redundant module. The hardware complexity ($A_{triple-rotation}$) of the unified micro-rotation of the triple-rotation CORDIC method is explained in Equation (4.2).

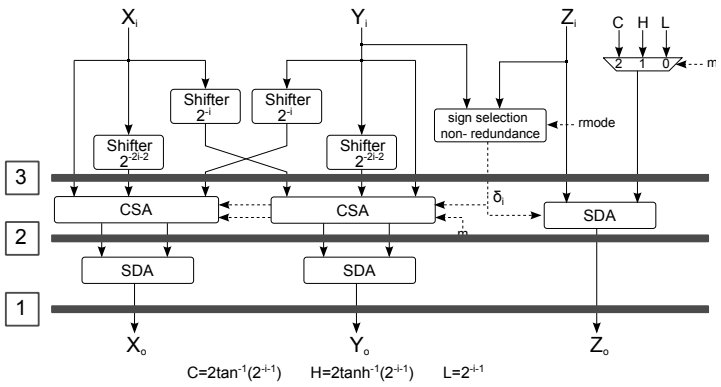
$$\begin{aligned} A_{double-rotation} &= 4 \cdot A_{shift} + 2 \cdot A_{CSA} \\ &\quad +3 \cdot A_{SDA} + A_{sign-sel} + A_{3-input\,Mux} \end{aligned} \tag{4.1}$$



(a) 1-stage

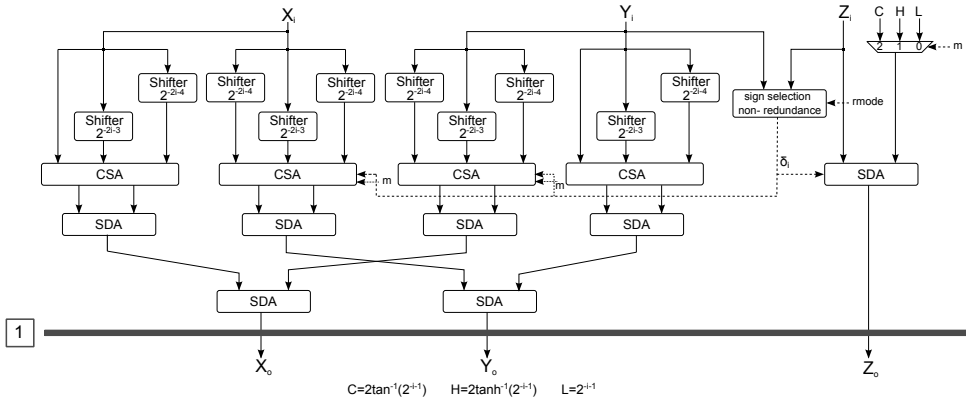


(b) 2-stage

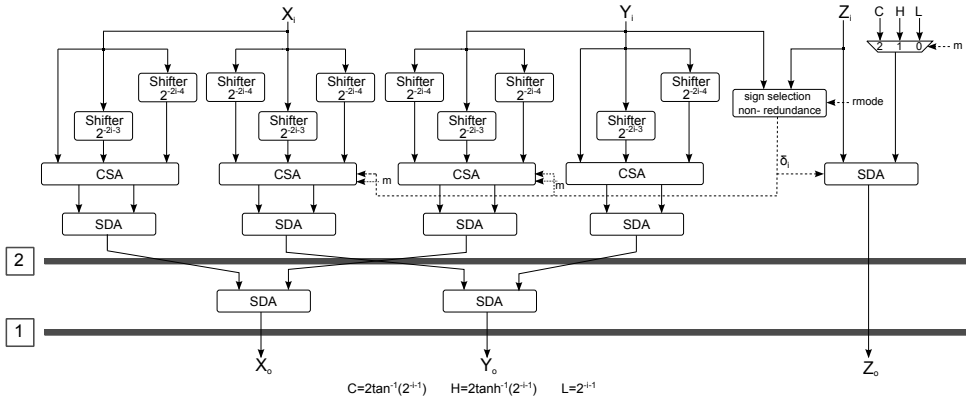


(c) 3-stage

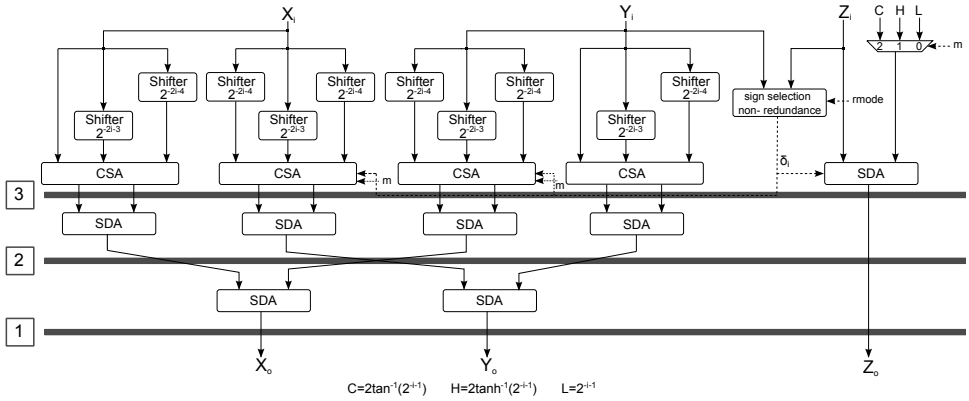
Fig. 4.2: The unified micro-rotation architecture of the double-rotation CORDIC method.



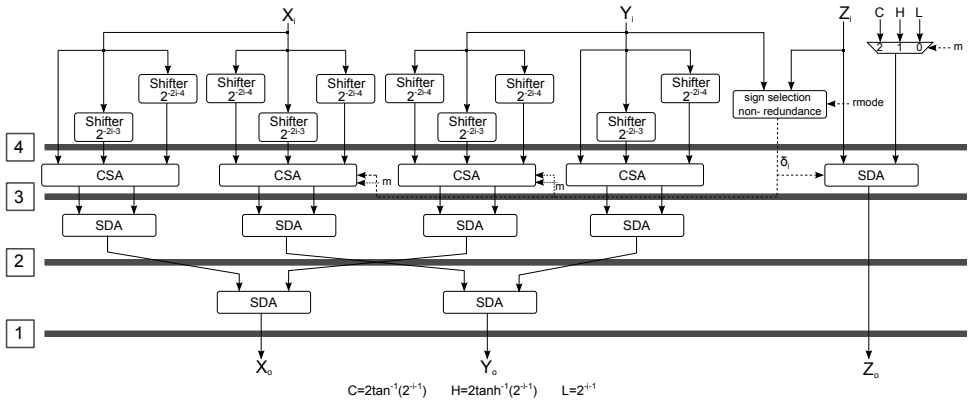
(a) 1-stage



(b) 2-stage



(c) 3-stage



(d) 4-stage

Fig. 4.3: The unified micro-rotation architecture of the triple-rotation CORDIC method.

Tab. 4.1: Synthesized results of the micro-rotation of the CORDIC methods on the Xilinx Virtex5 vlx110t-2ff1738 FPGA.

CORDIC	Utilization			Max. Freq. (MHz)
	Slice Reg.	Slice LUT	LUT-FF	
1-stage CV	252	0	0	213.668
2-stage CV	258	343	231	327.836
3-stage CV	273	355	239	340.513
1-stage DR	663	0	0	129.653
2-stage DR	258	698	230	198.210
3-stage DR	746	1,228	405	243.496
1-stage TR	1,582	0	0	101.811
2-stage TR	258	1,704	252	139.581
3-stage TR	547	1,775	488	193.581
4-stage TR	970	1522	739	243.496

CV = Conventional; DR = Double-rotation; TR = Triple-rotation

$$\begin{aligned} A_{triple-rotation} &= 10 \cdot A_{shift} + 4 \cdot A_{CSA} \\ &\quad +7 \cdot A_{SDA} + A_{sign-sel} + A_{3-input\,Mux} \end{aligned} \tag{4.2}$$

4.2.2 Resource Consumption and Performance Analysis

The performance and resource comparisons of the unified micro-instruction of the double-rotation (*DR*) and triple-rotation (*TR*) methods to the conventional (*CV*) method in the several pipeline architectures are presented in Tab. 4.1. The architecture is modelled with *VHDL* based on the 32-bit fixed-point format, where the integer and fraction parts are 8-bit and 24-bit respectively. The models are synthesized on the targeted FPGA, Xilinx Virtex5 vlx110t-2ff1738 device to investigate their performance and efficiency. The resource utilizations of the three architectures are increased according to the step up of pipeline stages. At the same stage, the logic area of the conventional method is smaller than the double-rotation and triple-rotation methods due to a lower computational complexity. The two methods can provide higher time efficiency than the conventional one.

4.3 A High Precision CORDIC Core

4.3.1 Algorithm

The main characteristics of the double-rotation and triple-rotation CORDIC methods are: (1) a reduction of computational latency, and (2) high computational accuracy with the small number of iterations. Thus, in this section such characteristics are applied to design a high accuracy CORDIC core algorithm. The algorithm runs the double-rotation method in normal-accuracy mode, and executes the triple-rotation method in high-accuracy mode. It consists of three main parts, i.e. pre-processing, CORDIC processing, and post-processing as shown in Algorithm 13.

Tab. 4.2 is simultaneously considered with the Algorithm 13 due to parameter relationship. The functional parameter (*func*) is in accordance with the function number (*No.*) shown in the table. Function numbers 1, 2 and 3 are used to perform trigonometric functions such as sine, cosine, polar-to-rectangular functions. Function numbers 4, 5 and 6 are used to build hyperbolic functions such as sine and cosine hyperbolic functions. Function numbers 7 and 8 are finally used to implement linear functions, i.e. the multiplication and division functions, respectively.

On the pre-processing step, the computational mode (*hs*) is determined in either normal-accuracy mode or high-accuracy mode, where the start (*start*) and end (*end*) indexes will be set up. Afterwards, the input parameters x_{start}, y_{start}, and z_{start} will be initialized corresponding to the *start* and *func* parameters. Within the CORDIC processing step, the unified micro-rotation of either the double-rotation method or the triple-rotation method will be executed in the pipeline. The execution in either the rotation mode or vectoring mode and the coordinate systems depends on the rotating parameter *rmode* and the coordinate parameter *m*. Finally, the post-processing step will compensate the computed results (if necessary) with inversion of the constant scaling factor corresponding to *func* in Tab. 4.2.

4.3.2 Computational Time Investigation

A block diagram of the high accuracy CORDIC core according to Algorithm 4.2 is shown in Figure 4.4a. The input variables, i.e. x_i, y_i, and z_i, will be adjusted by the convergence extension module. Then, the module will provide the adapted variables (x_{in}, y_{in}, z_{in}) and the configuration parameters (*hs*, *Iter*, *K*, K^{-1}, *func*, *rmode*) corresponding to the required functionality and the selected CORDIC mode. Generally, there are two types of the convergence extension methods that are introduced to solve the convergence range problem as described in section 3.9, i.e. the mathematic identity method [114] [139] and the sequential index expansion method [50]. The mathematic identity method applies the mathematic properties such as trigonometric or hyperbolic identities to compress inputs x_i, y_i, and z_i, and to decompress outputs x_{out}, y_{out}, and z_{out} generated by the high accuracy

Algorithm 13 [x_{out}, y_{out}, z_{out}] = High-ACC-CORDIC(x_{in}, y_{in}, z_{in}, $Iter$, K, K^{-1}, $rmode$, m, $func$, hs)

{******** Pre-processing ********}
[x_{start}, y_{start}, z_{start}]= PRE-PROCESSING-CORDIC(hs, $Iter$, K, $func$, x_i, y_i, z_i)
{******** CORDIC processing ********}
for $i = start$ to end **do**
    if $(rmode = 0)$ **then**
        if $(z_i \geq 0)$ **then**
            $\delta_i = 1$
        else
            $\delta_i = -1$
        end if
    else
        if $(y_i \geq 0)$ **then**
            $\delta_i = -1$
        else
            $\delta_i = 1$
        end if
    end if
    if $(hs = 1)$ **then**
        $x_{i+1} = x_i \cdot (1 - m \cdot (2^{-2i-3} + 2^{-2i-4})) - \delta_i(m \cdot (2^{-i-1} + 2^{-i-2}) - 2^{-3i-6}) \cdot y_i$
        $y_{i+1} = \delta_i \cdot x_i \cdot (2^{-i-1} + 2^{-i-2} - m \cdot 2^{-3i-6}) + (1 - m \cdot (2^{-2i-3} + 2^{-2i-4})) \cdot y_i$
        if $(m = 0)$ **then**
            $z_{i+1} = z_i - \delta_i \cdot 3 \cdot 2^{-i-2}$
        else
            $z_{i+1} = z_i - \delta_i \cdot 3 \cdot tan^{-1}(2^{-i-2})$
        end if
    else
        $x_{i+1} = x_i - m \cdot (\delta_i \cdot 2^{-i} \cdot y_i + 2^{-2i-2} \cdot x_i)$
        $y_{i+1} = y_i + \delta_i \cdot 2^{-i} \cdot x_i - m \cdot 2^{-2i-2} \cdot y_i$
        if $(m = 0)$ **then**
            $z_{i+1} = z_i - \delta_i \cdot 2^{-i}$
        else
            $z_{i+1} = z_i - \delta_i \cdot 2 \cdot tan^{-1}(2^{-i-1})$
        end if
    end if
    $x_i = x_{i+1}$
    $y_i = x_{i+1}$
    $z_i = z_{i+1}$
end for
{******** Post-processing ********}
x_{out}, y_{out}, z_{out}] = POST-PROCESSING-CORDIC($rmode$, $func$, K^{-1}, x_i, y_i, z_i)
return x_{out}, y_{out}, z_{out}

Algorithm 14 [x_{start}, y_{start}, z_{start}] = PRE-PROCESSING-CORDIC(hs, $Iter$, K, $func$, x_i, y_i, z_i)

{******** Pre-processing ********}
if $(hs = 1)$ **then**
    $start = 2$, $end = Iter + 1$
else
    $start = 1$, $end = Iter$
end if
if $(func = 1)$ or $(func = 4)$ **then**
    x_{start}=K, y_{start}=0, z_{start}=z_i
else
    x_{start}=x_i, y_{start}=y_i, z_{start}=z_i
end if

Algorithm 15 [x_{out}, y_{out}, z_{out}] = POST-PROCESSING-CORDIC($rmode$, $func$, K^{-1}, x_i, y_i, z_i)

{******** Post-processing ********}
if $(rmode = 0)$ **then**
    if $(func = 2)$ or $(func = 5)$ **then**
        $x_{out} = x_i \cdot K^{-1}$
        $y_{out} = y_i \cdot K^{-1}$
        $z_{out} = z_i$
    else
        $x_{out} = x_i$
        $y_{out} = y_i$
        $z_{out} = z_i$
    end if
else
    $x_{out} = x_i \cdot K^{-1}$
    $y_{out} = y_i$
    $z_{out} = z_i$
end if

Tab. 4.2: The relationship of the elementary functions performed by the high precision CORDIC and all input arguments.

No.	Functions	$rmode$	Initial values			Configuration values		
			x_{in}	y_{in}	z_{in}	K^{-1}		m
						$hs=0$	$hs=1$	
Circular/Trigonometric Coordinate System								
1	$x_{out} = cos(z_{in})$ $y_{out} = sin(z_{in})$	0	K	0	given value	0	0	1
2	$x_{out} = x_{in} \cdot cos(z_{in}) - y_{in} \cdot sin(z_{in})$ $y_{out} = x_{in} \cdot sin(z_{in}) + y_{in} \cdot cos(z_{in})$		given value	given value	given value	K_{dr}^{-1}	K_{tr}^{-1}	1
3	$x_{out} = \sqrt{x_{in}^2 + y_{in}^2}$ $z_{out} = z_{in} + tan^{-1}\left(\frac{y_{in}}{x_{in}}\right)$	1	given value	given value	given value	K_{dr}^{-1}	K_{tr}^{-1}	1
Hyperbolic Coordinate System								
4	$x_{out} = cosh(z_{in})$ $y_{out} = sinh(z_{in})$	0	K	0	given value	0	0	-1
5	$x_{out} = x_{in} \cdot cosh(z_{in}) + y_{in} \cdot sinh(z_{in})$ $y_{out} = x_{in} \cdot sinh(z_{in}) + y_{in} \cdot cosh(z_{in})$		given value	given value	given value	K_{dr}^{-1}	K_{tr}^{-1}	-1
6	$x_{out} = \sqrt{x_{in}^2 - y_{in}^2}$ $z_{out} = z_{in} + tanh^{-1}\left(\frac{y_{in}}{x_{in}}\right)$	1	given value	given value	given value	K_{dr}^{-1}	K_{tr}^{-1}	-1
Linear Coordinate System								
7	$x_{out} = x_{in}$ $y_{out} = y_{in} + x_{in} \cdot z_{in}$	0	given value	given value	given value	1	1	0
8	$x_{out} = x_{in}$ $z_{out} = z_{in} + \frac{y_{in}}{x_{in}}$	1	given value	given value	given value	1	1	0

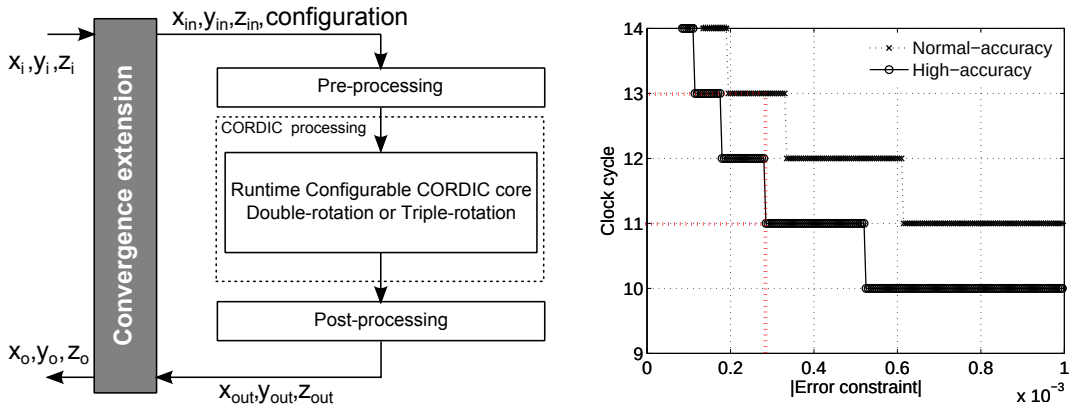


Fig. 4.4: The block diagram of the high precision CORDIC core with the convergence extension module and its computational latency.

CORDIC core.

Fig. 4.4a illustrates the computational latency of the normal-accuracy and high accuracy modes(double-rotation and triple-rotation). Thus, the computational time investigation of the high accuracy CORDIC core including the convergence extension module can be examined as follows:

1) Suppose that T_{ext}, T_{pre}, T_{dr}, T_{tr}, and T_{post} are internal delay of the convergence extension, the pre-processing, the double-rotation, the triple-rotation and the post-processing, respectively.

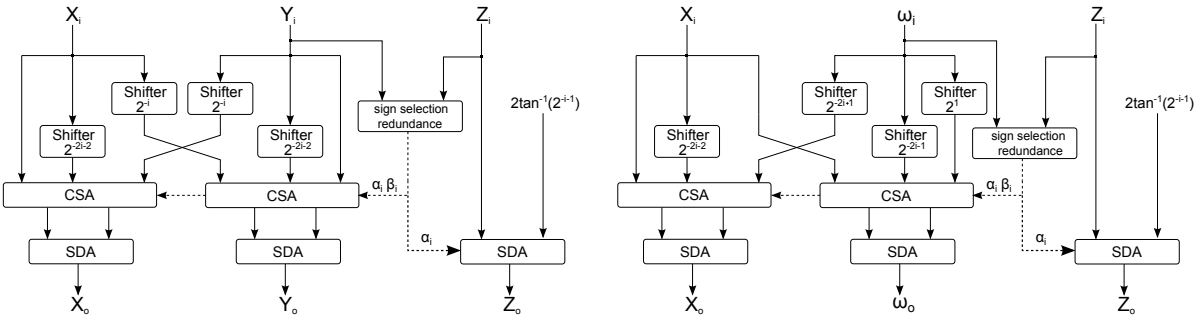
2) T_{dr} and T_{tr} depend on the number of CORDIC iterations ($N_{iter-dr}$, $N_{iter-tr}$) corresponding to the expected accuracy as shown in Fig. 3.3.

The computational latency of the normal-accuracy (T_{dr}) and high-accuracy (T_{tr}) modes are expressed as:

$$\begin{aligned} T_{dr} &= T_{micro-dr} \cdot N_{iter-dr} \\ T_{tr} &= T_{micro-tr} \cdot N_{iter-tr}, \end{aligned} \tag{4.3}$$

where $T_{micro-dr}$ and $T_{micro-tr}$ are the computational delays of the micro-rotation of the double-rotation and triple-rotation methods. The number of iterations is denoted as $N_{iter-dr}$ and $N_{iter-tr}$. Therefore, the computational delay investigation of the high accuracy CORDIC core with the convergence extension module can be described as:

$$\begin{aligned} T &= 2 \cdot T_{ext} + T_{pre} + T_{CORDIC} + T_{post}, \\ T_{CORDIC} &= \begin{cases} T_{dr} & hs = 0 \ (Normal\ accuracy\ mode), \\ T_{tr} & hs = 1 \ (High\ accuracy\ mode) \end{cases} \end{aligned} \tag{4.4}$$



(a) Double-rotation [114] (b) 2D-Householder [49]

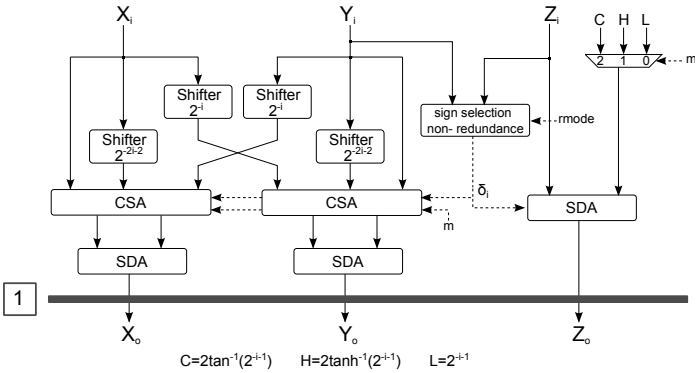
Fig. 4.5: The existing constant scaling factor CORDIC methods based on the redundant method.

Assume that T_{ext}, T_{pre}, T_{post}, $T_{micro-dr}$, and $T_{micro-tr}$ are equal to 1 clock cycle, then the delays of the high accuracy CORDIC core based on the double-rotation and triple-rotation methods with at 3.0E-4 of the expected absolute error are at 11 clock cycles and 13 clock cycles. Fig. 4.4b shows the trade-off between the expected absolute computational error and the delays of the high accuracy CORDIC Core in normal-accuracy mode and high-accuracy mode with the absolute errors ranging from 0 to 1.0E-3.

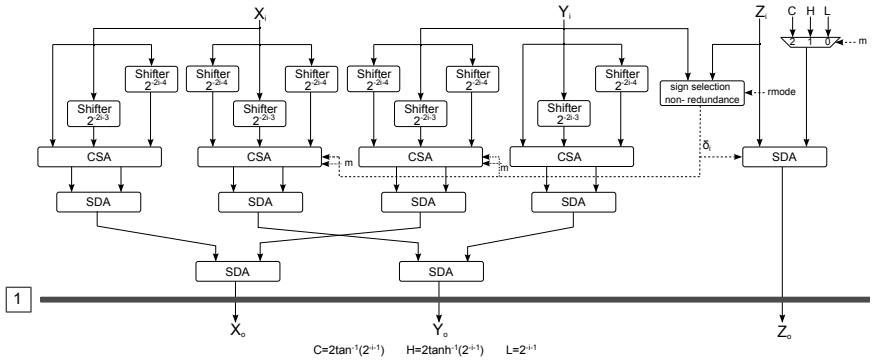
4.3.3 Performance Comparison

This section compares the proposed CORDIC methods with the existing ones. The micro-rotation in pipelined (unfolded) digit-parallel architecture has been brought up for consideration in speed and area performance, where pre-processing and post-processing units are ignored. Basic components normally used to implement the CORDIC consist of 3-to-2 carry-save-adder (*CSA*), sign-digit-adder (*SDA*), redundant sign selection (*SIGN-SEL*), non-redundant sign selection (*SIGN-SEL-NON*), and right shifter (*SHR*). The basic components are implemented and analysed in various data width at 16-bit, 32-bit, and 64-bit on the 90-nm Faraday silicon technology. The synthesis results are individually normalized based on the delay and consumed area of the targeted technology; they are then normalized again in the various data widths as presented in Tab. 4.3.

The two existing constant scaling factor CORDIC methods, i.e. the redundant double-rotation [114] CORDIC and the 2D-Householder *2D-Householder* [49] CORDIC, have been put forward for comparison with the proposed CORDIC methods, whose architectures on rotation mode in the circular coordinate system are illustrated in Figs. 4.5 and 4.6. Tab. 4.4 compares the speed and area performance of these CORDIC methods. Based on the pipeline architecture, the computational operators corresponding to each CORDIC algorithm are performed as the delay model *Delay*. Also the number of utilized basic computational operators, conforming to Figs. 4.5 and 4.6, is modelled as the area models. In [114], the redundant double-rotation method with a constant scaling factor is applied to only the CORDIC in rotation mode. The proposed double-rotation method extends to vectoring mode. The redundant rotation direction ($\alpha_i, \beta_i \in \{-1, 0, 1\}$) are employed



(a) Proposed double-rotation



(b) Proposed triple-rotation

Fig. 4.6: The proposed constant scaling factor CORDIC methods based on the non-redundant method.

Tab. 4.3: Basic components synthesis results on the 90-nm Faraday silicon technology.

Basic component	16-bit		32-bit		64-bit	
	D(ns)	A(μm^2)	D(ns)	A(μm^2)	D(ns)	A(μm^2)
3-to-2 CSA	0.13/0.0807	252.23/0.5708	0.13/0.0458	504.11/0.4836	0.13/0.0245	1,005.87/0.3978
SDA	1.61/1	442.96/1	2.84/1	806.74/0.7714	5.31/1	1,534.29/0.6068
SHR	0.70/0.4348	377.89/0.8533	0.62/0.2183	1,045.85/1	0.92/0.1733	2,528.40/1
SIGN-SEL	0.15/0.0932	13.33/0.0301	0.15/0.0528	13.33/0.0127	0.15/0.0282	13.33/0.0053
SIGN-SEL-NON	0.01/0.0063	2.35/0.0053	0.01/0.0035	2.35/0.0220	0.01/0.0019	2.35/0.0009

Tab. 4.4: The time and area performance of the CORDIC methods in the pipeline (unfolded) digit-parallel architecture.

Double-rotation [114]	Delay	$y_i + \beta_i 2^{-2i-2} x_i - \alpha_i 2^i y_i \Longrightarrow D_{SHR} + D_{SIGN-SEL} + D_{CSA} + D_{SDA}$
	Area	$4 \cdot A_{SHR} + 2 \cdot A_{CSA} + 3 \cdot A_{SDA} + A_{SIGN-SEL}$
2D-Householder [49]	Delay	$2 \cdot \omega_i - \beta_i 2^{-2i-1} \omega_i - \alpha_i 2^i x_i \Longrightarrow D_{SHR} + D_{SIGN-SEL} + D_{CSA} + D_{SDA}$
	Area	$4 \cdot A_{SHR} + 2 \cdot A_{CSA} + 3 \cdot A_{SDA} + A_{SIGN-SEL}$
Double-rotation	Delay	$x_i + \delta_i 2^{-i} y_i - \delta_i 2^{-2i-2} x_i \Longrightarrow D_{SHR} + D_{SIGN-SEL-NON} + D_{CSA} + D_{SDA}$
	Area	$4 \cdot A_{SHR} + 2 \cdot A_{CSA} + 3 \cdot A_{SDA} + A_{SIGN-SEL-NON}$
Triple-rotation	Delay	$(x_i - 2^{-2i-3} x_i - 2^{-2i-4} x_i) - (\delta_i 2^{-i-1} y_i + \delta_i 2^{-i-2} y_i - \delta_i 2^{-3i-6} y_i)$ $\Longrightarrow D_{SHR} + D_{SIGN-SEL-NON} + D_{CSA} + 2 \cdot D_{SDA}$
	Area	$10 \cdot A_{SHR} + 4 \cdot A_{CSA} + 7 \cdot A_{SDA} + A_{SIGN-SEL-NON}$

in [114], but this dissertation apply the non-redundant rotation direction to the double-rotation CORDIC method. From the delay and area models in Tab. 4.4, the delay and consumed area on the 16-bit data width of the double-rotation CORDIC method of [114] can be evaluated as follows:*Delay* : $0.4348 + 0.0932 + 0.0807 + 1$=*1.6087*, *Area* : $4 \times 0.8533 + 2 \times 0.5708 + 3 + 0.0301$=*7.5849*.

In [49], the redundant *2D-Householder* CORDIC method is applied on both the rotation mode and vectoring mode. Its scaling factor is performed by the on-line computation, increasing complexity for *VLSI* implementation, however, for the sake of simplification, the on-line computation is neglected in this comparison. The delay and area on the 16-bit data width of the *2D-Householder* CORDIC method can be estimated as follows: *Delay* : $0.4348 + 0.0932 + 0.0807 + 1$=*1.6087*, *Area* : $4 \times 0.8533 + 2 \times 0.5708 + 3 + 0.0301$=*7.5849*. By the same method the delay and area on the 16-bit of the proposed double-rotation CORDIC method can be evaluated as follows : *Delay* : $0.4348 + 0.0063 + 0.0807 + 1$=*1.5218*, *Area* : $4 \times 0.8533 + 2 \times 0.5708 + 3 + 0.0053$=*7.5601*, and *Delay* : $0.4348 + 0.0063 + 0.0807 + 2$=*2.5218*, *Area* : $10 \times 0.8533 + 4 \times 0.5708 + 7 + 0.0053$=*17.822* for the proposed triple-rotation CORDIC method. The time and area efficiency of these CORDIC methods in different data-widths can be illustrated in Tab. 4.5.

From the comparison, the proposed non-redundant double-rotation CORDIC method

Tab. 4.5: Normalized speed and area performance comparison of the proposed CORDIC methods and the existing CORDIC methods in different data width.

CORDIC methods	16-bit		32-bit		64-bit	
	Delay	Area	Delay	Area	Delay	Area
Double-rotation [114]	1.6087	7.5856	1.317	7.2916	1.226	6.6214
2D-Householder [49]	1.6087	7.5856	1.317	7.2916	1.226	6.6214
Proposed double-rotation	1.5217	7.5608	1.2677	7.2811	1.1996	6.6171
Proposed triple-rotation	2.5218	17.822	2.2678	17.332	2.1996	15.84

provides better time efficiency than the redundant double-rotation and *2D-Householder* CORDIC methods. In addition, the proposed double-rotation method is extended to vectoring mode and also unemployed to use the on-line constant scaling factor computation. The area efficiency of the three CORDIC methods has similar values because they use the same number of basic components. The proposed triple-rotation CORDIC methods are also evaluated to demonstrate time and area efficiencies. Although the time and area efficiency of the triple-rotation CORDIC method are lower than the double-rotation CORDIC methods, but the method provides higher computational accuracy. The time efficiency can be improved by increasing a pipeline stage or by enhancing the performance of the adder. However, the trade-off between time and area efficiencies has to be examined.

4.4 Data Conversion

The data representation problem between a floating-point unit and a fixed-point unit can be eliminated by data conversion. The data conversion converts data in floating-point format to fixed-point format, and vice versa. The conversion is required because the standard and non-standard operational units proposed in chapter 2 are designed in floating-point format whereas the elementary functional unit based on the CORDIC methods is designed in fixed-point format. Therefore, to synchronize the two units, the floating-to-fixed algorithm and fixed-to-floating algorithm are introduced in this section.

4.4.1 Fixed-Point Representation

The fixed-point number is useful for representing fractional values after the range of the values has been evaluated and properly estimated. Fig. 4.7 illustrates the fixed-point bit-representation (n) which is composed of two main parts, i.e. the integer-bit part (QI) and the fractional-bit part (QF).

Actually, there are two types of fixed-point representation, i.e. signed-magnitude format and two's complement format. In signed-magnitude format, the most significant bit (MSB) of QI is defined as sign-bit, where it is set for negative number representation. On

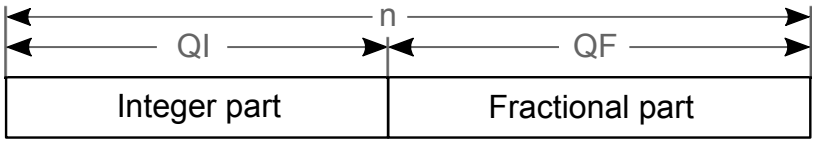


Fig. 4.7: Fixed-point format

the other hand, *QI* is two's complement format is represented with a negative number.

4.4.2 Floating-to-Fixed Algorithm

Since there are two types of fixed-point representation, i.e. signed magnitude and 2's complement, the *Floating-to-Fixed algorithms* on signed-magnitude format and on two's complement format are introduced in Algorithms 17 and 18. From the two algorithms, a common function is the *MANTISSA-ADJUST* function. It is used for comparison and adjustment of the bit-width of mantissa in floating-point format to be the same as the bit width of the fraction of a fixed-point format.

Algorithm 16 MANTISSA-ADJUST

Require: $mantissa_{old}, nf, QF$
Ensure: $frac$
1: $diff = QF - nf$
2: $temp = b'1 \parallel mantissa_{old}$
3: **if** $diff \geq 0$ **then**
4: $\quad frac = SHL(temp, QF - nf)$
5: **else**
6: $\quad frac = SHR(temp, nf - QF)$
7: **end if**
8: **return** $frac$

The algorithm of the *MANTISSA-ADJUST* function is depicted in Algorithm 16, where the input parameters ($mantissa_{old}$, *nf*, *QF*) are the mantissa's input of a floating-point number, the bit-width of mantissa and fraction respectively. *frac* is an output fraction in fixed-point format.

The Algorithm 17 requires five input arguments, i.e. a floating-point input X_{float}, the number of floating-point exponents *ne*, the number of floating-point mantissas *nf*, the number of fixed-point integer-bit *QI*, the number of fixed-point factional-bit *QF*. The algorithm is considered as illustrated by the following steps.

- *Extraction*: Sign bit, exponent, and mantissa of the floating-point input will be mapped to variable *Sign*, *Exp*, and *Mantissa* depending on the determined *nf* and *OF*.
- *Comparison*: The variable Exp_{fixed} will be performed by subtraction of the current floating-point exponent value with the maximum value of the floating-point format (2^{ne-1}) when *Exp* is equal to or gather than $2^{ne-1}-1$. Otherwise, the variable Exp_{fixed}

will be the subtracting result of $2^{ne-1}-1$ and the current floating-point exponent value.

- *Computation*: In this step, the value $Y_{fixed-signed}$ and overflow flag OV will be calculated by first investigating the Exp_{fixed} and $2^{ne-1}-1$. If the Exp_{fixed} is greater than or equal to $2^{ne-1}-1$, then $Y_{fixed-signed}$ will be $h'FFF\cdots$ and OV is set. Otherwise, the integer-bit and fractional-bit will be performed by a shift right function *SHR* corresponding to variables *Mantissa* and Exp_{fixed}, whereas OV is zero.

Algorithm 17 *Floating-to-Fixed* based on signed magnitude format

Require: $X_{float}, ne, nf, QI, QF$
Ensure: $Y_{fixed-signed}, OV$
1: {******** Extraction ********}
2: $Sign = X_{float}[ne+nf]$
3: $Exp = X_{float}[ne+nf-1:nf]$
4: $Mantissa = MANTISSA - ADJUST(X_{float}[nf-1:0]), hf, QF$
5: {******** Comparison ********}
6: **if** $Exp \geq (2^{(ne-1)}-1)$ **then**
7: $\quad Exp_{fixed} = X_{float}[ne+nf-1:nf] - 2^{(ne-1)}$
8: **else**
9: $\quad Exp_{fixed} = (2^{(ne-1)}-1) - X_{float}[ne+nf-1:nf]$
10: **end if**
11: {******** Computation ********}
12: **if** $Exp_{fixed} \geq (2^{(ne-1)}-1)$ **then**
13: $\quad Y_{fixed-signed}[QI+QF-1:0] = h'FFF\cdots$
14: $\quad OV = 1$
15: **else**
16: $\quad Y_{fixed-signed}[QI+QF-2:0] = SHR(Mantissa, Exp_{fixed})$
17: $\quad$ **if** $Sign = 0$ **then**
18: $\quad\quad Y_{fixed-signed}[QI+QF-1] = 0$
19: $\quad$ **else**
20: $\quad\quad Y_{fixed-signed}[QI+QF-1] = 1$
21: $\quad$ **end if**
22: $\quad OV = 0$
23: **end if**
24: **return** $Y_{fixed-signed}, OV$

Algorithm 18 explains the data conversion from floating-point to fixed-point in the 2's complement format. There are also three steps for the conversion. *Extraction* and *Comparison* steps in Algorithm 18 look like the same process as in Algorithm 17. The difference is the Computation step, where the variable $Y_{fixed-2CMP}$ will be 2's complement as shown in Line 19.

Algorithm 18 *Floating-to-Fixed* based on 2's complement format

Require: $X_{float}, ne, nf, QI, QF$
Ensure: $Y_{fixed-2CMP}, OV$
1: {******** Extraction ********}
2: $Sign = X_{float}[ne + nf]$
3: $Exp = X_{float}[ne + nf - 1 : nf]$
4: $Mantissa = MANTISSA - ADJUST(X_{float}[nf - 1 : 0], hf, QF)$
5: {******** Comparison ********}
6: **if** $Exp \geq (2^{(ne-1)} - 1)$ **then**
7: $\quad Exp_{fixed} = X_{float}[ne + nf - 1 : nf] - 2^{(ne-1)}$
8: **else**
9: $\quad Exp_{fixed} = (2^{(ne-1)} - 1) - X_{float}[ne + nf - 1 : nf]$
10: **end if**
11: {******** Computation ********}
12: **if** $Exp_{fixed} \geq (2^{(ne-1)} - 1)$ **then**
13: $\quad Y_{fixed-2CMP}[QI + QF - 1 : 0] = h'FFF \cdots$
14: $\quad OV = 1$
15: **else**
16: $\quad Y_{fixed-2CMP}[QI + QF - 1 : 0] = SHR(Mantissa, Exp_{fixed})$
17: $\quad$ **if** $Sign = 1$ **then**
18: $\quad\quad Y_{fixed-2CMP} = 2CMP(Y_{fixed-2CMP})$
19: $\quad\quad$ {****** 2CMP is a two's complement operation ******}
20: $\quad$ **end if**
21: $\quad OV = 0$
22: **end if**
23: **return** $Y_{fixed-2CMP}, OV$

Tab. 4.6 presents the computational results, where the input data is converted from floating-point format to fixed-point format on both the signed-magnitude (Algorithm 17) and the 2's complement representations (Algorithm 18).

4.4.3 Fixed-to-Floating Algorithm

Similarly, Algorithm 19 presents the *Fixed-to-Float* Algorithm based on signed magnitude format. Five input arguments are required, i.e. a fixed-signed input $X_{fixed-signed}$, ne, nf, QI, and QF. The algorithm shows four steps as in the following points.

- *Sign detection*: based on signed magnitude format, the *MSB* of $X_{fixed-signed}$ will be checked. If *MSB* equals to 1, then *Sign* will be set 1; otherwise 0;
- *LOD computation*: the function *LOD* proposed in section 2.3.1 will be applied, where a variable *Position* obtains the leading one position.
- *Mantissa computation*: the mantissa is performed in this step by either right shifting (*SHR*) or left shifting (*SHL*), where the number of shifting depends on the *Position* and *QF*.
- *Concatenation*: the variable *Exp* is calculated and the variables *Sign*, *Exp*, and *Mantissa* will be packed.

Like the *Fixed-to-Float Algorithm* on the signed magnitude format, the *Fixed-to-Float Algorithm* is based on the 2's complement format consisting of four processing points is shown in Algorithm 20. The *Sign Detection* is different from the *Fixed-to-Float algorithm* in Algorithm 19, where the variable *Sign* is investigated in 2's complement form. Tab. 4.7 presents the computational results, where the input data is converted from fixed-point format to floating-point format for both signed-magnitude(Algorithm 19) representation and 2's complement representation (Algorithm 20).

The floating-point to fixed-point data conversion algorithms in Algorithms 17 and 19 and the fixed-point and floating-point data conversion algorithms in Algorithms 19 and 20 are modeled and verified by *VHDL*, where the conversion time is in one clock cycle. The models are synthesized on the Xilinx Virtex5 vlx110t-2ff1738 FPGA in order to investigated their performance and efficiency. Their synthesis results are shown in Tab. 4.8. The floating-to-fixed module provides better time efficiency than the fixed-to-floating module due to the lowest latency. In order to improve the time efficiency of the fixed-to-floating module, the pipeline technique can be applied.

Tab. 4.6: Executional example of the *Floating-to-Fixed algorithm* based on signed magnitude and 2's complement conversion, where ne and nf equal to 8-bit and 23-bit as well as QI and QF are equal to 8-bit and 24-bit.

Real number	X_{float}	$Sign$	Exp_{fixed}	$Mantissa$	$Y_{fixed-signed}$	$Y_{fixed-2CMP}$	OV
0.025	$h'3CCCCCCC$	$b'0$	-6	$b'1.10011001100110011001100$	$h'00066666$	$h'00066666$	$b'0$
-0.025	$h'BCCCCCCC$	$b'1$	-6	$b'1.10011001100110011001100$	$h'80066666$	$h'FFF9999A$	$b'0$
0.25	$h'3E800000$	$b'0$	-2	$b'1.00000000000000000000000$	$h'00400000$	$h'00400000$	$b'0$
-0.25	$h'BE800000$	$b'1$	-2	$b'1.00000000000000000000000$	$h'80400000$	$h'FFC00000$	$b'0$
1.25	$h'3FA00000$	$b'0$	0	$b'1.01000000000000000000000$	$h'01400000$	$h'01400000$	$b'0$
-1.25	$h'BFA00000$	$b'1$	0	$b'1.01000000000000000000000$	$h'81400000$	$h'FEC00000$	$b'0$
50.0	$h'42480000$	$b'0$	5	$b'1.10010000000000000000000$	$h'32000000$	$h'32000000$	$b'0$
-50.0	$h'C2480000$	$b'1$	5	$b'1.10010000000000000000000$	$h'B2000000$	$h'CE000000$	$b'0$
100.0	$h'42C80000$	$b'0$	6	$b'1.10010000000000000000000$	$h'64000000$	$h'64000000$	$b'0$
-100.0	$h'C2C80000$	$b'1$	6	$b'1.10010000000000000000000$	$h'E4000000$	$h'9C000000$	$b'0$
200.0	$h'43480000$	$b'0$	7	$b'1.10010000000000000000000$	$h'FFFFFFFF$	$h'FFFFFFFF$	$b'1$
-200.0	$h'C3480000$	$b'1$	7	$b'1.10010000000000000000000$	$h'FFFFFFFF$	$h'FFFFFFFF$	$b'1$

$Y_{fixed-signed}$: a fixed-point value in signed-magnitude format, $Y_{fixed-2CMP}$: a fixed-point value in two's complement format

Algorithm 19 *Fixed-to-Float* based on signed magnitude format

Require: $X_{fixed-signed}, ne, nf, QI, QF$
Ensure: $Y_{float-signed}$
1: {****** Sign detection ******}
2: **if** $X_{fixed-signed}[QI + QF - 1] = 1$ **then**
3: $Sign = 1$
4: **else**
5: $Sign = 0$
6: **end if**
7: {****** LOD computation ******}
8: $Position = LOD(X_{fixed-signed}[QI + QF - 2 : 0])$
9: {****** Mantissa computation ******}
10: $Temp = X_{fixed-signed}[QI + QF - 2 : 0]$
11: **if** $(Position \geq QF)$ **then**
12: $Mantissa = SHR(Temp, Position - QF + 1)$
13: **else**
14: $Mantissa = SHL(Temp, QF - Position - 1)$
15: **end if**
16: {****** Concatenation ******}
17: $Exp = (2^{(ne-1)} - 1) - QF + Position$
18: $Y_{float} = Sign \parallel Exp \parallel Mantissa$
19: **return** Y_{float}

Algorithm 20 *Fixed-to-Float* based on 2's complement format

Require: $X_{fixed-2CMP}, ne, nf, QI, QF$
Ensure: $Y_{float-2CMP}$
{****** Sign Detection ******}
if $X_{fixed-2CMP}[QI + QF - 1] = 1$ **then**
    $Sign = 1$
    $Temp_{fixed} = 2CMP(X_{fixed-2CMP})$
else
    $Sign = 0$
    $Temp_{fixed} = X_{fixed-2CMP}$
end if
{****** LOD Computation ******}
$Position = LOD(Temp_{fixed}[QI + QF - 2 : 0])$
{****** Mantissa Computation ******}
$Temp = Temp_{fixed}[QI + QF - 2 : 0]$
if $(Position \geq QF)$ **then**
    $Mantissa = SHR(Temp, Position - QF + 1)$
else
    $Mantissa = SHL(Temp, QF - Position - 1)$
end if
{****** Concatenation ******}
$Exponent = (2^{(ne-1)} - 1) - QF + Position$
$Y_{float} = Sign \parallel Exp \parallel Mantissa$
return Y_{float}

Tab. 4.7: Executional example of the *Fixed-to-Floating algorithm* in signed magnitude and 2's complement formats, where *ne* and *nf* equal to 8-bit and 23-bit as well as *QI* and *QF* are equal to 8-bit and 24-bit.

$X_{fixed-signed}$	$X_{fixed-2CMP}$	$Sign$	$Exponent$	$Mantissa$	Y_{float}	Real number
$h'0006666$	$h'00066666$	$b'0$	$b'01111001$	$b'1.10011001100110011001100$	$h'3CCCCCCC$	0.025
$h'8006666$	$h'FFF9999A$	$b'1$	$b'01111001$	$b'1.10011001100110011001100$	$h'BCCCCCCC$	-0.025
$h'0040000$	$h'00400000$	$b'0$	$b'01111101$	$b'1.00000000000000000000000$	$h'3E800000$	0.25
$h'8040000$	$h'FFC00000$	$b'1$	$b'01111101$	$b'1.00000000000000000000000$	$h'BE800000$	-0.25
$h'0140000$	$h'01400000$	$b'0$	$b'01111111$	$b'1.01000000000000000000000$	$h'3FA00000$	1.25
$h'8140000$	$h'FEC00000$	$b'1$	$b'01111111$	$b'1.01000000000000000000000$	$h'BFA00000$	-1.25
$h'3200000$	$h'32000000$	$b'0$	$b'10000100$	$b'1.10010000000000000000000$	$h'424800000$	50.0
$h'B200000$	$h'CE000000$	$b'1$	$b'10000100$	$b'1.10010000000000000000000$	$h'C24800000$	-50.0
$h'6400000$	$h'64000000$	$b'0$	$b'10000101$	$b'1.10010000000000000000000$	$h'42C800000$	100.0
$h'E400000$	$h'9C000000$	$b'1$	$b'10000101$	$b'1.10010000000000000000000$	$h'C2C800000$	-100.0

$Y_{float-signed}$: a floating-point value in signed-magnitude format, $Y_{float-2CMP}$: a floating-point value in two's complement format

Tab. 4.8: Synthesized results of VHDL implementation of the floating-to-fixed and fixed-to-floating modules on the Xilinx Virtex5 vlx110t-2ff1738 FPGA.

Algorithm	Utilization			Max. Freq. (MHz)
	Slice Reg.	Slice LUT	LUT-FF	
Floating-to-Fixed	63	561	32	222.547
Fixed-to-Floating	64	326	31	141.153

4.5 Design and Architecture of a Arithmetic Accelerator

Actually, a floating-point arithmetic accelerator is designed to accelerate the computation of a main processor, and to support a classic processor, where a floating-point arithmetic unit is unemployed. It is very helpful to reduce the design cost and the time-to-market duration of the hardware processing system platform based on the classical processor implementation. Like the developed version of the accelerator proposed in this section 2.5. A CORDIC arithmetic unit is included to perform elementary functions. The stalling input instruction is also managed by the handshaking method whereas *ready* signals are applied for data synchronization between a receiver module and a sender module.

4.5.1 Design and Architecture

The design and architecture of the accelerator is conceived to support a bus communication system, where two bus interfaces are employed to interconnect between a main processor and the proposed arithmetic accelerator. The architecture of the proposed arithmetic accelerator can be illustrated in Fig. 4.8. The architecture has five arithmetic units, i.e. *ADD*, *MUL*, *SoP*, *PoS*, CORDIC, to perform the addition/subtraction, multiplication, sum-of-product, product-of-sum functions, and elementary functions such as sine, cosine, sine hyperbolic, cosine hyperbolic, etc. All arithmetic units interconnect via an internal bus system consisting of an internal output-bus and an *internal input-bus*. The multiplexer (*MUX*) units, *C1*, *C2*, *C3*, and *C4*, are employed to multiplex a computational results performed by the five arithmetic units to write on the *internal output-bus*. The *internal input-bus* consists of five bus-lines, i.e. *op1*, *op2*, *op3*, *ic*, and bi_{rdy}. Similar to the responsibility of the multiplexer units, the switch unit *C5* is applied to control the writing of computational results from the CORDIC unit onto the *internal output-bus*. The internal output-bus also consists of five bus-lines, i.e. *opr1*, *opr2*, *opr3*, *oc*, and bo_{rdy}.

4.5.1.1 Micro-Instruction Set

Since the design of the floating-point arithmetic accelerator depends on the data-width of the bus system, the 32-bit data-width is applied in this case. The micro-instruction set of the proposed accelerator is designed based on a register-to-register concept, where

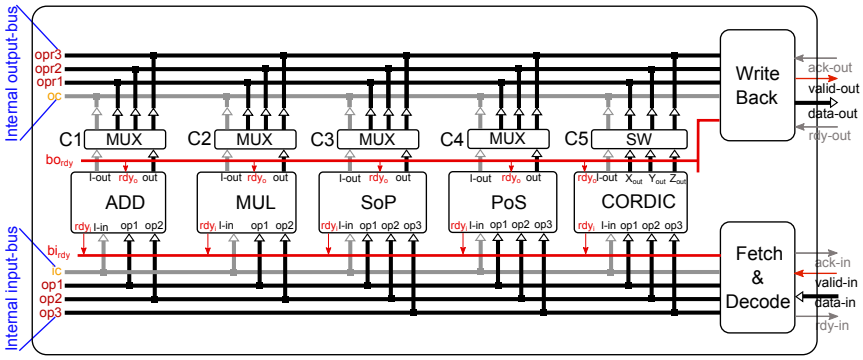


Fig. 4.8: The architecture of the floating-point arithmetic accelerator based on the CORDIC unit.

input and output operands will be held in registers op_1, op_2, op_3, R_1, and R_3, respectively. The 15 micro-instructions are provided for the proposed arithmetic accelerator as shown in Tab. 4.9. There are two and three input operands for the micro-instruction, i.e. op_1, op_2, and op_3, where the computational results of an executing instruction are either one word or two words depending on such instructions. Fig. 4.9 presents short and long instruction formats, *#F1* and *#F2*, as well as short and long replay formats, *#S1* and *#S2*, of the proposed floating-point arithmetic accelerator.

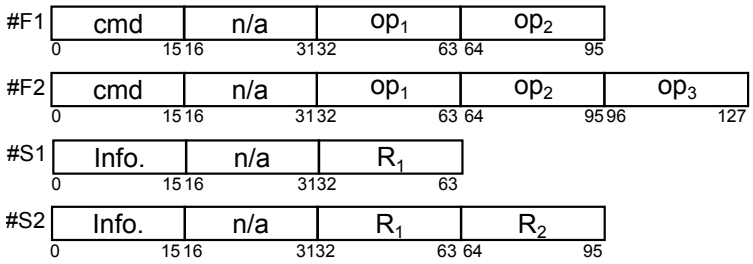


Fig. 4.9: Instruction format *#F1* and *#F2* as well as reply format #S1 and *#S2* of the floating-point arithmetic accelerator

4.5.1.2 A Fetch-and-Decode Unit

This unit is responsible for receiving and decoding an intermediate instruction from outside; next a control word and information corresponding to the intermediate instruction's property is created to manipulate related components during the computational process. The two stage machines, i.e. fetch stage and decode stage, will be executed. The fetch stage will fetch the input instruction from the external bus system to issue the processor with the instruction. The number of fetching instructions is examined from *Cmd* which is the first word of the instruction. If *Cmd* equals to X'0001, X'0002, or X'000D, then the number of fetching instructions will be set 3 times, otherwise 4 times. Fig. 4.10 presents the timing diagram of the *Fetch-and-Decode* unit for the short instruction format *#F1* and the long instruction format *#F2*. There are four signals applied for fetching an intermediate instruction from the external bus system, i.e. *valid-in* signal, *data-in* signal, *rdy-in*

Tab. 4.9: The micro-instruction of the proposed floating-point arithmetic accelerator.

Cmd	Mnemonic	Operand	Operation	Description
x'0001	ADD	op_1, op_2, R_1	$R_1 \leftarrow op_1 + op_2$	Addition function
x'0002	MUL	op_1, op_2, R_1	$R_1 \leftarrow op_1 \cdot op_2$	Multiplication function
x'0003	PoS	op_1, op_2, op_3, R_1	$R_1 \leftarrow (op_1 + op_2) \cdot op_3$	Product-of-Sum function
x'0004	SoP	op_1, op_2, op_3, R_1	$R_1 \leftarrow (op_1 \cdot op_2) + op_3$	Sum-of-Product function
x'0005	SIN-COS	op_1, op_2, op_3, R_1, R_2	$R_1 \leftarrow cos(op_3)$ $R_2 \leftarrow sin(op_3)$	Cosine function Sine function
x'0006	SUM-SIN-COS	op_1, op_2, op_3, R_1, R_2	$R_1 \leftarrow op_1 \cdot cos(op_3) - op_2 \cdot sin(op_3)$ $R_2 \leftarrow op_1 \cdot sin(op_3) + op_2 \cdot cos(op_3)$	Subtraction of multiplication of Sine and Cosine function Addition of multiplication of Sin and Cos
x'0007	POLAR-REC	op_1, op_2, op_3, R_1, R_2	$R_1 \leftarrow \sqrt{op_1^2 + op_2^2}$ $R_2 \leftarrow op_3 + tan^{-1}\left(\frac{op_2}{op_1}\right)$	Polar to rectangular function
x'0008	SINH-COSH	op_1, op_2, op_3, R_1, R_2	$R_1 \leftarrow cosh(op_3)$ $R_2 \leftarrow sinh(op_3)$	Hyperbolic Cosine function Hyperbolic Sine function
x'0009	SUM-SINH-COSH	op_1, op_2, op_3, R_1, R_2	$R_1 \leftarrow op_1 \cdot cosh(op_3) + op_2 \cdot sinh(op_3)$ $R_2 \leftarrow op_1 \cdot sinh(op_3) + op_2 \cdot cosh(op_3)$	Addition of multiplication of hyperbolic Sin and Cosine function Addition of multiplication of hyperbolic Sin and Cosine function
x'000A	VEC-HYPERXY	op_1, op_2, op_3, R_1, R_2	$R_1 \leftarrow \sqrt{op_1^2 - op_2^2}$ $R_2 \leftarrow op_3 + tanh^{-1}\left(\frac{op_2}{op_1}\right)$	square root of difference of two constant values Hyperbolic arch tan with constant addition function
x'000B	ROT-LINEAR	op_1, op_2, op_3, R_1, R_2	$R_1 \leftarrow op_1$ $R_2 \leftarrow op_2 + (op_1 \cdot op_3)$	Sum-of-Product function corresponding to CORDIC boundary
x'000C	VEC-LINEAR	op_1, op_2, op_3, R_1, R_2	$R_1 \leftarrow op_1$ $R_2 \leftarrow op_3 + \frac{op_2}{op_1}$	Sum-of-Division function corresponding to CORDIC boundary
x'000D	DIV	op_1, op_2, R_1	$R_1 \leftarrow \frac{op_2}{op_1}$	Division function
x'000E	Ln	op_1, op_2, op_3, R_1	$R_1 \leftarrow Ln(op_3) = 2 \cdot tan^{-1}\left(\frac{op_2}{op_1}\right)$ $op_1 = op_3 + 1, op_2 = op_3 - 1$	National logarithmic function
x'000F	SQRT	op_1, op_2, op_3, R_1	$R_1 \leftarrow \sqrt{op_3} = \sqrt{op_1^2 - op_2^2}$ $op_1 = op_3 + 0.25, op_2 = op_3 - 0.25$	Square-root function

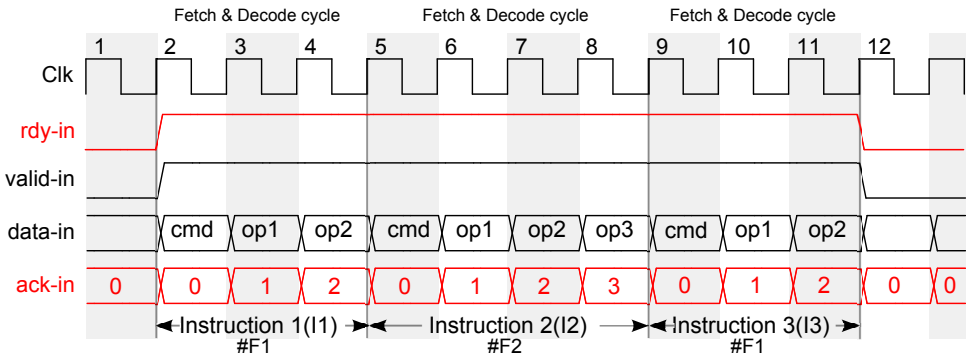


Fig. 4.10: Timing diagram of the *Fetch-and-Decode* unit for short instruction format *#F1* and long instruction format *#F2*.

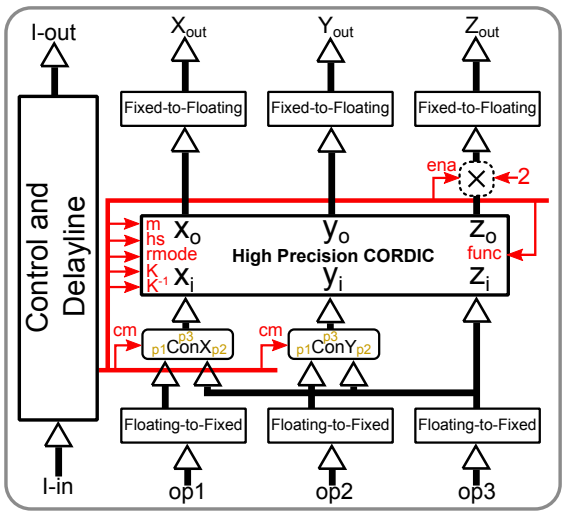


Fig. 4.11: The architecture of a CORDIC Unit

signal, and *ack-in* signal. As soon as *rdy-in* signal is active, the *valid-in* signal and the *data-in* signal are simultaneously detected and fetched. After the two signals have been presented to the *Fetch-and-Decode* unit, the *ack-in* value will be plus one in order to inform to the source of such instruction that the presented word has been already obtained by the processor. When the instruction is completely fetched, the value of the *ack-in* signal will be reset.

4.5.1.3 A CORDIC Unit

The architecture of the CORDIC unit illustrated in Fig. 4.11 is designed to cover the functionalities exhibited in Tab. 4.9. The architecture consists of six components, i.e. *Floating-to-Fixed*, *Fixed-to-Floating*, *ConX*, *ConY*, *Constant Multiplier*, and a high precision CORDIC components. These components are explained as follows.

- *Floating-to-fixed and fixed-to-floating components*: The components transform input data in floating-point to fixed-point format, and vice versa. The algorithms em-

ployed for the two data converters are described in section 4.4.

- *Control and delay-line component*: The component receives the information from the *I-in* signal and generates the control signals *ena* and *cm* in order to prepare the pre-processing and post-processing steps for natural logarithm and square root functions. Finally, the control signals will be propagated as depicted in Algorithm 21.
- *ConX and ConY components*: The *cm* signal generated by the control and delay-line component is detected with these components, where the cm signal is applied to manipulate inputs p_1 and p_2. Their functionality can be expressed in Equations (4.5) and (4.6).
- *Constant Multiplier component*: Whenever the *ena* signal is enabled, output Z_0 of the high accuracy CORDIC module is multiplied by two; otherwise they will be forwarded without being processed.
- *High accuracy CORDIC component*: This component is designed conforming to Algorithm 13. The component consists of six input parameters, i.e. *m*, *hs*, *rmod*, *K*, K^{-1} and *func*. The values of these parameters depend on the current function as illustrated in Tabs. 4.2 and 4.9. The parameter *func* is customized to conform to the parameter *cmd* in Tab. 4.10

Equation of ConX module

$$p_3 = \begin{cases} p_2 + 1.0 & if\,(CM = 1) \\ p_2 + 0.25 & if\,(CM = 2) \\ p_1 & Otherwise \end{cases} \tag{4.5}$$

Equation of ConY module

$$p_3 = \begin{cases} p_2 - 1.0 & if\,(CM = 1) \\ p_2 - 0.25 & if\,(CM = 2) \\ p_1 & Otherwise \end{cases} \tag{4.6}$$

4.5.1.4 A WriteBack Unit

The *WriteBack* unit is responsible for managing the computational results that are generated from the arithmetic units. The computational results are presented on the internal output-bus comprising of five buses, i.e. *op1*, *op2*, *op3*, *oc*, and bo_{rdy} as illustrated in Fig. 4.8. The computational results from all the arithmetic units are selected by an impartial policy such as a fairness arbiter mechanism, and arranged conforming to the reply format either #S1 or #S2 depending on an instruction *cmd*. The issue of the computational results from

Tab. 4.10: Mapping between the instruction *cmd* in Tab. 4.10 and the functional number *func* in Tab. 4.2.

Instruction *cmd*	Functional number *func*
x′0006	1
x′0007	2
x′0008	3
x′0009	4
x′000A	5
x′000B	6
x′000C	7
x′000D	8
x′000E	6
x′000F	6

Algorithm 21 Pre-Post Processing

Require: Cmd
Ensure: ena, cm
1: **if** $Cmd = x'000E$ **then**
2: $ena = b'1$
3: $cm = b'01$
4: **else if** $Cmd = x'000F$ **then**
5: $ena = b'0$
6: $cm = b'10$;
7: **else**
8: $ena = b'0$;
9: $cm = b'00$;
10: **end if**
11: **return** ena, cm

Tab. 4.11: Accuracy analysis of hardware's double-rotation CORDIC in various fixed-point representations.

Fixed-point format (QI:QF)	$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
2:29	3.6899E-5	3.3357E-5	3.5746E-5	7.8789E-7
4:27	3.6823E-5	3.3272E-5	3.5657E-5	7.8853E-7
6:25	3.6637E-5	3.2987E-5	3.5335E-5	7.8742E-7
8:23	3.5883E-5	3.1127E-5	3.3809E-5	9.1425E-7
10:21	3.7630E-5	2.4561E-5	3.7630E-5	2.5011E-6
12:19	4.9646E-5	5.1370E-7	4.9646E-5	9.1294E-6
14:17	1.2279E-4	3.1866E-7	1.2279E-4	2.7748E-5
16:15	4.7996E-2	4.6672E-4	4.7996E-2	1.4112E-2

Tab. 4.12: Accuracy analysis of hardware's triple-rotation CORDIC in various fixed-point representations.

Fixed-point format (QI:QF)	$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.\,Dev.\|error\|$
2:29	3.0749E-5	2.7798E-5	2.9788E-5	6.5657E-7
4:27	3.0686E-5	2.7727E-5	2.9714E-5	6.5711E-7
6:25	3.0531E-5	2.7489E-5	2.9446E-5	6.5618E-7
8:23	2.9903E-5	2.5940E-5	2.8174E-5	7.6187E-7
10:21	3.1358E-5	2.0468E-5	3.1358E-5	2.0843E-6
12:19	4.1372E-5	4.2808E-7	4.1372E-5	7.6078E-6
14:17	1.0232E-4	2.6555E-7	1.0232E-4	2.3123E-5
16:15	3.9997E-2	3.8893E-4	3.9997E-2	1.1760E-2

each arithmetic unit is controlled by a *ready-output* signal rdy_o via bo_{rdy} bus authorized by the impartial policy of this unit. The *ready-output* signal rdy_o of each arithmetic unit connects to the ready-input signal rdy_i internally in order to stall the fetching instruction of the *Fetch-and-Decode* unit. Therefore, there is no loss, duplication, or collision of data and instruction.

Fig. 4.12 exhibits the timing diagram of the *WritBack* unit. The computational results generated from the instruction numbers 1, 3, and 2 appear at clock number 9, 16 and 21 respectively. Reading these computational results is handled by bo_{rdy} signals number 1 to 5. As soon as a bo_{rdy} signal is active, the value of *opr* bus numbers 1 to 3 and their valid signal will read into the unit.

4.5.2 Implementation and Performance Analysis

According to the timing diagram in Fig. 4.10 and 4.12, the minimum computational delay comes from the floating-point adder/subtractor unit or the floating-point multiplier unit, where their pipeline stage is equal to 5 stages in order to maintain the maximum clock frequency at 100 MHz. Since the addition/subtraction and multiplication instructions are presented in the form of a short instruction format which consumes 3 clock cycles for fetching and decoding the instructions; by the same token, to issue the computational result generated by the unit into the external bus system, 2 clock cycles are dedicated for the *WriteBack* unit. Thus, the minimum computational latency of the floating-point arithmetic accelerator is at *10* clock cycles. The maximum computational delay depends on the number of iterations of the CORDIC unit (L_{Iter}). The number of iterations of the CORDIC unit N_{Iter} is derived from $L_{floating-to-fixed} + L_{inter.cordic} + L_{fixed-to-floating}$, where $L_{floating-to-fixed}$ is a latency of the Floating-to-Fixed unit, $L_{floating-to-fixed}$ is a latency of the Fixed-to-Floating unit, and $L_{inter.cordic}$ is the latency of the high precision CORDIC unit. The instruction of the computational CORDIC unit is presented in a form of long

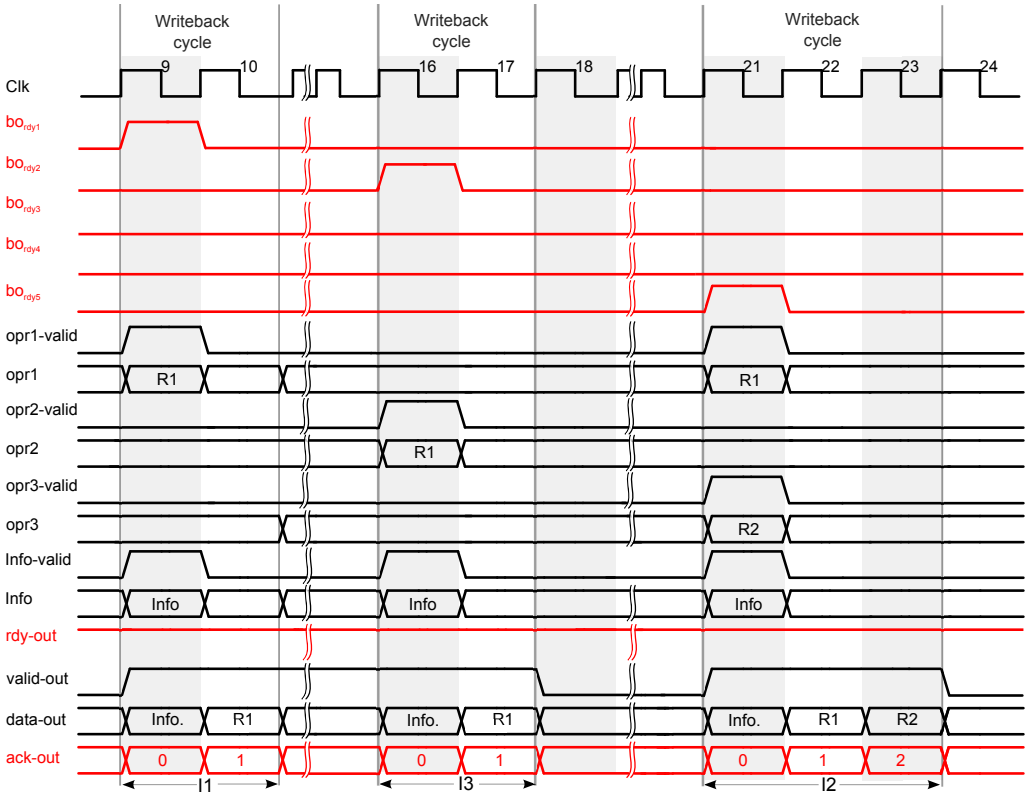


Fig. 4.12: Timing diagram of *Writeback* unit of the floating-point arithmetic accelerator

Tab. 4.13: Synthesis results using the 130-nm CMOS standard-cell technology from Faraday technology with target frequency at 500 MHz.

Measurements	Synthesis result
Number of slice registers	$0.6645\,\mu m^2$
Slack time (critical path)	$1.5\,ns$

instruction format, where 4 clock cycles are spent for fetching and decoding the instruction. By the same token, to issue the computational result generated by the unit into the external bus system, 3 clock cycles are dedicated for the *WriteBack* unit. Therefore, the maximum computational latency of the floating-point arithmetic accelerator are at $7 + L_{floating-to-fixed} + L_{inter.cordic} + L_{fixed-to-floating} = 9 + L_{inter.cordic}$ clock cycles. In this design, $L_{inter.cordic}$ is equal to 16 clock cycle, so the maximum computational latency is 25 clock cycles. The floating-point arithmetic accelerator has been synthesized with the 130-nm CMOS standard-cell technology library. In the architecture, an adder, a multiplier, a product-of-sum, a sum-of-product, a CORDIC core, a *Fetch-and-Decode*, and *WriteBack* units have been included in the current architecture.

Tab. 4.13 exhibits the synthesis results of the floating-point arithmetic accelerator by using the 130-nm CMOS standard-cell library from Faraday Technology Corporation. The target working frequency is at 500 MHz, resulting in a slack time (critical path of the core) of about 1.92 ns. From the synthesis results using the 130-nm CMOS technology,

Tab. 4.14: Synthesis results using the Xilinx Virtex 5 device xc5vlx110t-3ff-1136 FPGA.

	Utilisation	% of Total
Number of slice registers	24,423 out of 69,120	36%
Number of slice LUTs	5,046 out of 69,120	8%
Number of slice LUT-FF	92 out of 385	24%
Number of BUFG/BUFGCTRLs	1 out of 32	3%
Critical Delay	9.98 ns	
Maximum Frequency	100.02 MHz)	

the floating-point arithmetic accelerator clocks at 667 MHz, resulting in a minimum and maximum computation of about 2.67 *MFLOPS* and 6.67 *MFLOPS* (Mega Floating-Point Operation Per Second). Tab. 4.14 shows the synthesis results on the Xilinx Virtex 5 device xc5vlx110t-3ff-1136 FPGA, the maximum architecture clock rate is 100.02 MHz and consumes approximately 40% of the resource utilisation.

4.6 Design and Architecture of a Reconfigurable Streaming Processor

Streaming processors are commonly employed to accelerate the computations of a scientific formulas applied in many engineering and information technology applications [18] [100]. The scientific computations are specially required to solve some chemical and physical problems, signal and image processing problems and other problems in civil and mechanical engineering. The scientific computations are often very complex and sometimes also have very short time constraints. By using a general-purpose computer system, the scientific problem is functionally well solved. However, due to the complexity of the scientific computations, high-performance requirements of the computation cannot be met. Therefore, a specific streaming processor is proposed to solve the real time constraint problem. In this section, a reconfigurable streaming processor is proposed which is targeted for an adaptronic application [13] [93]. Although the proposed streaming processors focuses only on the hardware architecture and the core of the streaming processor, in the future, the core will also be interconnected through a 2D mesh on-chip network in order to gain higher performance and system reliability.

4.6.1 Design and Architecture

The design and architecture of the proposed streaming processor is illustrated in Fig. 4.13. There are five floating-point-based arithmetic units, i.e. an adder (*ADD*), a multiplier (*MUL*), a product-of-sum (*PoS*), a sum-of-product (*SoP*), a fast CORDIC core (CORDIC) units. Two memory units (*M1* and *M2*) and a shift register (*SREG*) connected to bus a system. The arithmetic and memory units are interconnected through 6 buses, i.e. *op1*

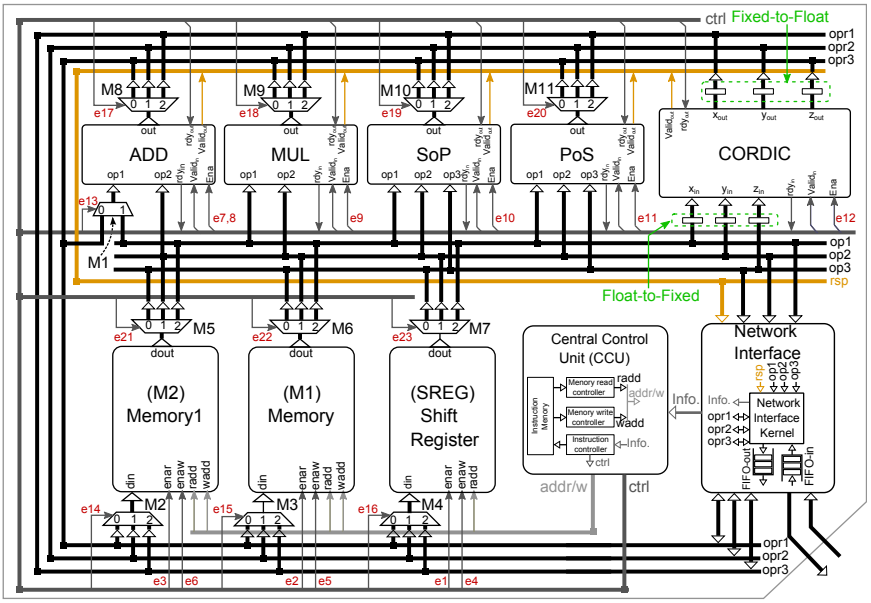


Fig. 4.13: The architecture of the floating-point streaming processor for adaptive digital control system.

bus, *op2* bus, *op3* bus, *opr1* bus, *opr2* bus, and *opr3* bus. The *op1*, *op2*, and *op3* buses are applied for operand 1, operand 2 and operand 3 of each arithmetic units and connected also to the output port of the memory units. The *opr1*, *opr2*, and *opr3* buses are employed for the operation results of the arithmetic units and united also to the input ports of the memories and the shift-register.

The central controller unit (*CCU*) is implemented into the streaming processor to control the processor configuration and the flow of streaming computations. The processor configuration and the flow of the streaming computations are manipulated by fetching and executing instruction vectors stored into the instruction memory and implemented on the CCU. Each instruction vector contains micro-codes to enable the arithmetic and memory units and to control the streaming computation flows. Another component that would be implemented into the streaming processor is an on-chip network interface (*OCNI*). The *OCNI* is utilized to plug the streaming processor onto a network-on-chip (*NoC*). The *OCNI* is applied to assemble and disassemble message before injecting and after ejecting messages to and from the NoC, respectively.

4.6.2 CORE Configuration, Micro-Instruction, and Timing Diagram

The streaming processor core runs a streaming floating-point operation by reading an instruction stored in the *CCU*. A set of streaming floating-point operations can be established to represent an adaptive signal processing algorithm [38] [93]. Hence, by writing a microprogram or a set of instructions stored in the *CCU*, the streaming processor are configured to run a specific adaptive signal processing algorithm.

Tab. 4.15: Control-bit (enable) signal for the floating-point arithmetic and memory units.

Ctrl. type	Enable bit	Comments
RM Ctrl	e1	enable read Shift-Register (SREG)
	e2	enable read Memory 1 (M1)
	e3	enable read Memory 2 (M2)
WM Ctrl	e4	enable write Shift-Register (SREG)
	e5	enable write Memory 1 (M1)
	e6	enable write Memory 2 (M2)
FPU Ctrl	e7	enable FP addition (ADD)
	e8	enable FP subtraction (ADD)
	e9	enable FP multiplication (MUL)
	e10	enable FP sum-of-product (SoP)
	e11	enable FP product-of-sum (PoS)
	e12	enable fast CORDIC core (CORDIC)
IOB Ctrl	e13	input selection for multiplexer 1 (M1)
	e14	input selection for multiplexer 2 (M2)
	e15	input selection for multiplexer 3 (M3)
	e16	input selection for multiplexer 4 (M4)
	e17	output selection for demultiplexer 5 (M5)
	e18	output selection for demultiplexer 6 (M6)
	e19	output selection for demultiplexer 7 (M7)
	e20	output selection for demultiplexer 8 (M8)
	e21	output selection for demultiplexer 9 (M9)
	e22	output selection for demultiplexer 10 (M10)
	e23	output selection for demultiplexer 11 (M11)

In general, one instruction consists of 6 control fields, i.e. *Read-Memory Control* (*RM Ctrl*), *Write-Memory Control* (*WM Ctrl*), *Floating-Point Unit Control* (*FPU Ctrl*), *Bus Input-Output Control* (*IOB Ctrl*), and *Streaming Counting Control* (*CNT Ctrl*). The *FPU Ctrl*, *RM Ctrl* and *WM Ctrl* control fields are utilized to enable the arithmetic operations of the floating-point units and read-write operations of the memory units. The *IOB Ctrl* control field is employed to enable the bus connections with the arithmetic and memory units. Tab. 4.15 describes the explanations of all enable-bit signals. In order to control the computational flow of a number of streaming data sets the *CNT Ctrl* field is introduced in the instruction format.

Fig. 4.6.2 illustrates the configuration of the streaming processor to perform the operation number 1 (data streaming multiplication). In this operation, data x_j from a shift register (*SREG*) and w_i from memory 1(*M1*) are read in parallel and uploaded into *op1* bus and *op2* bus, respectively. Since the multiplier (*MUL*) unit is enable, then multiplication

Tab. 4.16: List of floating-point operations for the adaptive *LMS* signal processing.

No.	Computation	Flop. Unit	$op1$ (from)	$op2$ (from)	$opr3$ (store in)
1	$a_j = w_j \times x_j$ $j \in \psi$	MUL	w_j (M1)	x_j (SREG)	a_j (M2)
2	$y = \sum_j a_j$ $j \in \psi$	ADD	a_j (Bus $opr3$)	next a_j (M2)	y_j (M2)
3	$e = d - y$	ADD (Sub.)	d (M1)	y (M2)	e (M2)
4	$b_j = e - \beta$	MUL	e (M2)	β (M1)	b_j (M2)
5	$c_j = x_j \times b_j$ $j \in \psi$	MUL	x_j (SREG)	b_j (M2)	c_j (M2)
6	$w_j = w_j + c_j$ $j \in \psi$	ADD	w_j (M1)	c_j (M2)	new w_j (M1)

Note: $x_j = x(k - j)$

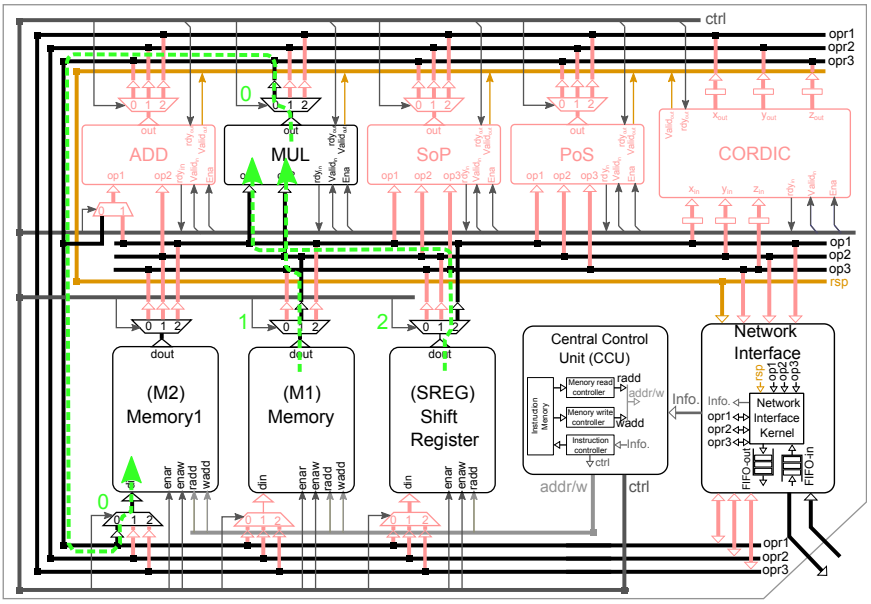


Fig. 4.14: Example of core configuration for the streaming computation number 1 according to Tab. 4.16.

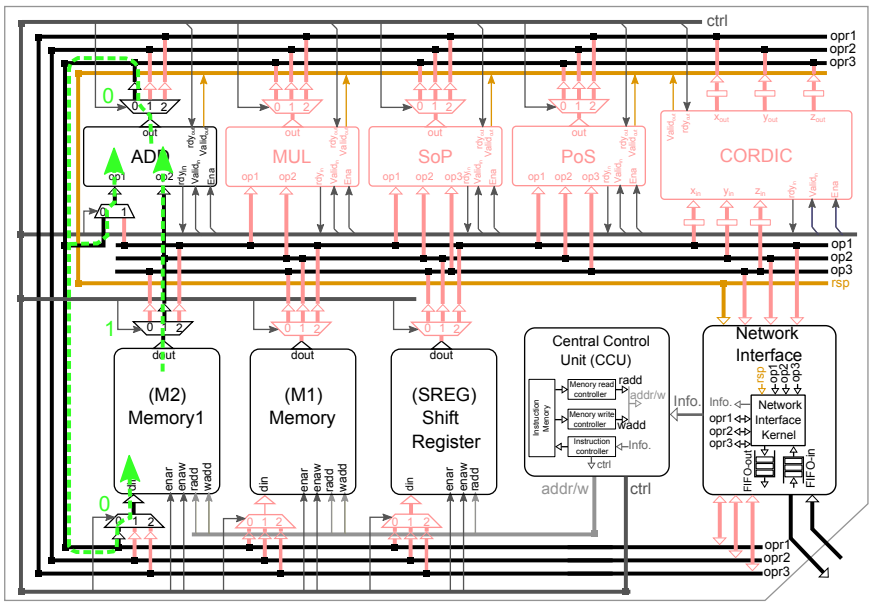


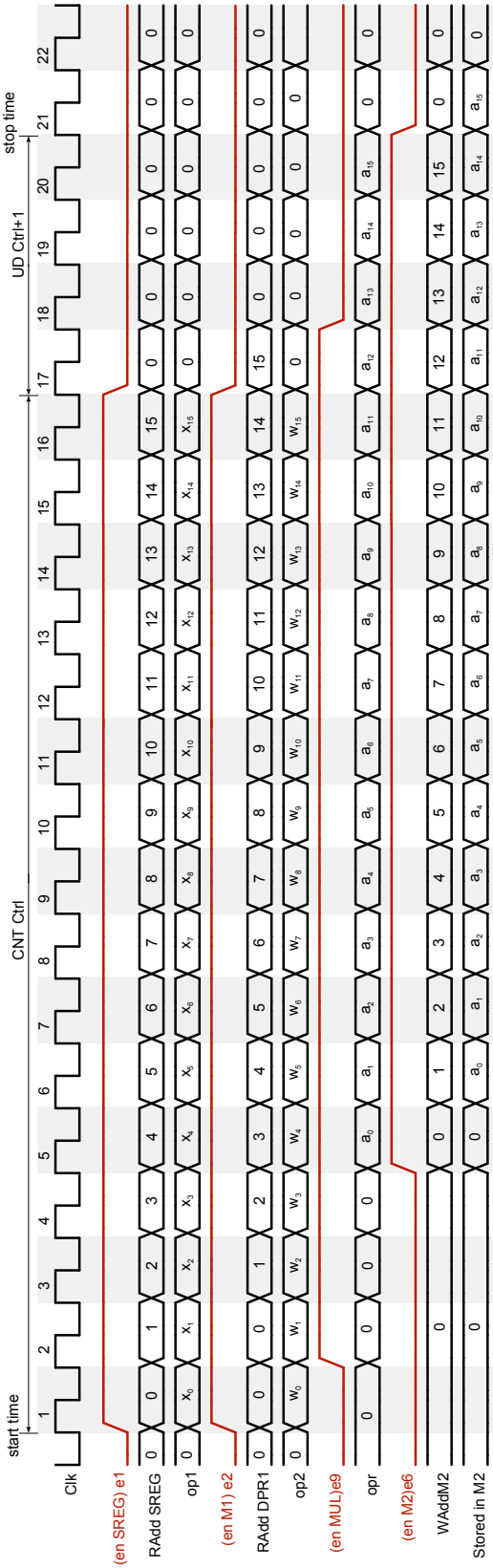
Fig. 4.15: Example of core configuration for the streaming computation number 2 according to Tab. 4.16.

of both data is made. The operation result a_j is then stored in memory 2 (*M2*). A number of N_{tap} pipeline operations are performed in this case. The timing diagram of the data stream multiplication shown in Fig. before is presented in Fig. 4.17. As presented in the figure, the 16 memory-to-memory streaming multiplicative computation is performed with the pipeline data path method. The computation is started by enabling the reading operation mode of the *SREG* and *M1*. Two data streams from both memory units flow from the input buses (*op1* and *op2* buses) to the output bus (*opr*) via the *MUL* arithmetic unit to perform the streaming multiplicative computation. The streaming computation results are finally stored (written) in the *M2*. The *CCU* controls the synchronization of the data reading and writing operations performed on the memory units.

Another processor configuration is depicted in Fig. 4.15. The figure represents the configuration to perform operation number 2 (data streaming accumulation). In this operation, the data a_j stored in the *M2* is uploaded onto bus *op2*. The multiplexer *M1* and the adder unit (*ADD*) are enabled in this case. The data is accumulated by performing an adding operation between operands on the bus *op2* and the bus *opr*. Finally, the operation results are stored again in the *M2* unit. Thus, during certain execution times, the *M2* performs concurrent read and write operations, i.e. uploading data onto the bus *op2* while storing data from the bus *opr*.

4.6.3 Implementation and Performance Analysis

The floating-point streaming processor has been synthesized with a 130-nm CMOS standard-cell technology library. In the architecture, an adder, a multiplier, a product-of-sum, a sum-of-product, two memories, a shift register as well as a CORDIC core unit have been



Tab. 4.17: Timing diagram of the pipeline streaming computation number 1

Tab. 4.18: Synthesis results using the 130-nm CMOS standard-cell technology from Faraday technology with target frequency 500 MHz.

Measurements	Synthesis result
Number of slice registers	$0.9867\,um^2$
Slack time (critical path)	$2.0\,ns$

Tab. 4.19: Synthesis result using the Xilinx Virtex 2 device xc2vp30-7-ff896 FPGA.

	Utilization	% of Total
Number of slice registers	5,752 of 13,696	42%
Number of bounded IOBs	212 of 556	38%
Number of BRAM	2 out of 136	1%
Number of GCLKs	2 out of 16	12%
Critical Delay	10.43 ns	
Maximum Frequency	95.86 MHz	

included in the current architecture. The size of the memories is 256×32, i.e. it has 256 data slots and each of them has a 32-bit data width. The shift register is set to have 128 data slots. The number of slots of the memory units can be reconfigured in the architecture.

Tab. 4.18 illustrates the synthesis results of the streaming processor by using the 130-nm CMOS standard-cell library from Faraday Technology Corporation. The target working frequency is 500 MHz, resulting in a slack time (critical path of the core) of about 1.92 ns. From the synthesis result using the 130-nm CMOS technology, the streaming processor can be clocked at 500 MHz, resulting in a streaming computation of about 5 *MFLOPS*. Compared to the Intel FPMAC processor (with 6.2-GFLOPs performance) [122], the performance of our streaming processor is lower, because the Intel FPMAC processor uses smaller technology (90-nm). Tab. 4.19 shows the synthesis result on the Xilinx Virtex 5 device xc5vlx110t-3ff-1136 FPGA, the architecture maximum clock rate can be clocked at a maximum of 95.86 MHz and consumes approximately 50% of the resource utilization.

4.7 Arithmetic Co-processor/Processor Comparison

Information of the published floating-point and fixed-point co-processor/processor is illustrated in this section. The information detail architecture, performance, and efficiency is all shown in Tab. 4.20. Tab.4.21 shows the list of floating-point co-processor/processor in the pipeline architecture and considers the computational performance in *FLOPS*.

Tab. 4.20: Specification of the floating-point and fixed-point co-processors.

Arithmetic Co-processor	Method	Function	Technology	Area/Resource	Max. Frequency	Architecture	Representation
SAGA [110]	CORDIC	all elementary functions	CMOS 1.2 nm	19 mm^2	25 MHz	Iterative	Fixed-Point
PSEUDEC [78]	CORDIC	all elementary functions	CMOS 1.0 μm	20 mm^2	75 MHz	Pipeline	Fixed-Point
Reconfig [91]	General proposed method	$C \leftarrow A \cdot B$	Xilinx	$\approx$ 6,024 Slices	25.5 MHz	Iterative	Fixed-Point
		$C \leftarrow \sqrt{A}$	unspecific Virtex				
		$C \leftarrow \frac{A}{B}$	model				
		$C \leftarrow A^2$					
Matrix [14]	General proposed method	$C \leftarrow A + B$	Xilinx Virtex4	$\approx$ 1,908 Slices	25 MHz	Iterative	Fixed-Point
		$C \leftarrow A^T$					
		$C \leftarrow Max(\lvert A_{i,j}\rvert, \cdots, \lvert A_{n-1,j}\rvert)$					
		$C \leftarrow O(A^3)$					
		$C \leftarrow A \cdot B$					
Molnar [79]	General proposed method	$C \leftarrow A + B$	CMOS 1.2 um	17,000 transistors	40 MHz	Pipeline	Floating-Point
Wei [128]	CORDIC in circular	$C \leftarrow Cos(A)$	Actel FPGA	no information	48 MHz	Pipeline	Fixed-Point
		$C \leftarrow Sin(A)$					
		$C \leftarrow tan(\frac{A}{B})$					
		$C \leftarrow \sqrt{A^2 + B^2}$					
Leeke [66]	CORDIC	all elementary functions	CMOS 0.25 um	6 mm^2	80 MHz	Pipeline	Fixed-point
Ju-Ho [55]	General purpose method	$C \leftarrow A \cdot B$	CMOS 0.18 μm	10.2 mm^2	200 MHz	Iterative	Fixed-point
		$C \leftarrow A \pm B$					
Hybrid [136]	CORDIC	all elementary functions	Xilinx Virtex5	22,189 Slice Reg.	173 MHz	Pipeline	Floating-point
				20,443 Slice LUT.			
Proposed accelerator	CORDIC and	all elementary functions	CMOS 130 nm	0.6645 mm^2	666.67 MHz	Pipeline	Floating-Point
	General purpose method	$C \leftarrow A \cdot B$	Xilinx Virtex5	24,423 Slice Reg.	100.02 MHz		
		$C \leftarrow A \pm B$		5,046 Slice LUT.			
		$D \leftarrow (A \cdot B) + C$		92 Slice LUT-FF.			
		$D \leftarrow (A + B) \cdot C$					

Tab. 4.21: Performance of the floating-point co-processors/processors in the pipeline architecture.

Co-processor/Processor	Max. Frequency	Computational Performance (FLOPS)
Proposed accelerator	667 MHz,	2.67-6.67 MFLOPS
Proposed streaming processor	500 MHz	5 MFLOPS
Molnar [79]	40 MHz	0.4 MFLOPS
Hybrid [136]	173 MHz	1.73 MFLOPS
Intel FPMAC processor [122]	6.2 GHz	6.2 GFLOPS

4.8 Summary

The design and architecture of the arithmetic unit consisting of the standard functional units, the non-standard functional units, and the elementary functional units are introduced and applied to the floating-point accelerator and the reconfigurable streaming floating-point processor. The summaries of this chapter conclude as follows.

- *The unified micro-rotations of the double-rotation and triple-rotation CORDIC methods are considered*. The design and architecture of the two unified micro-rotations are proposed in the pipeline, and investigated based on the basic components, i.e. shifters, *CSAs*, *SDAs*, and multiplexers. They are modelled, verified by *VHDL*, and synthesized based on the Xilinx Virtex5 FPGA. Their synthesis results are compared with the unified micro-rotation of the conventional method in terms of performance and efficiency.

- *The algorithm, design, and architecture of the high accuracy CORDIC core are introduced*. The algorithm applies the double-rotation method in the normal accuracy and the triple-rotation method for the high accuracy. The algorithm can be reconfigured to perform the CORDIC computation in the circular, hyperbolic, and linear coordinate systems by changing the configurable variables at runtime. The timing model, which can be used to investigate the computational latency of the high accuracy CORDIC core, is introduced. The comparison of the published CORDIC methods, i.e. the redundant double-rotation method and *2D-Hoseholder* method, with the proposed methods is described in the time and area models. The comparison results illustrate that the proposed non-redundant double-rotation methods provide better time efficiency than the redundant double-rotation and *2D-Householder*. In addition, the proposed double-rotation method is extended to vectoring mode and also unemployed to use on-line constant scaling factor computation. The area efficiencies of the three methods have similar values due to consuming the same number of basic components. The proposed triple-rotation is evaluated to demonstrate the time and area efficiencies. However, its time efficiency can be improved by increasing the pipeline stages or by enhancing the performance of the adder.

- *Two data conversion algorithms*, i.e. the *Floating-to-Fixed Algorithm* and the *Fixed-to-Floating Algorithm*, are proposed, where they are used to convert data from floating-point to fixed-point and fixed-point to floating-point.
- *The design and architecture of the accelerator and the reconfigurable streaming processor are introduced for the case studies*. The main purpose of the accelerator is to accelerate the computation of a main processor while the reconfigurable streaming processor is designed specifically for computing the streaming data. However, the two processors are designed to support the floating-point computation, where the basic operations, i.e. the adder/subtractor, the product-of-sum, and the sum-of-product, are performed in floating-point format, whereas the elementary functions built by CORDIC in fixed-point format.

Chapter 5

Verification on the Closed-Loop Control System for Heavy Ion Synchrotron Application

Contents

The beam phase and magnitude detector employed in the closed-loop control system for heavy ion synchrotron application is applied for verification of the proposed CORDIC methods. An overview of the closed-loop control system is described; afterwards algorithm, design, implementation, and simulation of the digital phase and magnitude detection module is discussed. The sequential index extension method is used for convergence range extension. The design of the digital phase and magnitude detector is modelled and simulated based on *VHDL* hardware description language. The simulation result based on the actual digital signals of the closed-loop control is compared with Matlab/Simulink ideal result in order to verify the proposed CORDIC's computation.

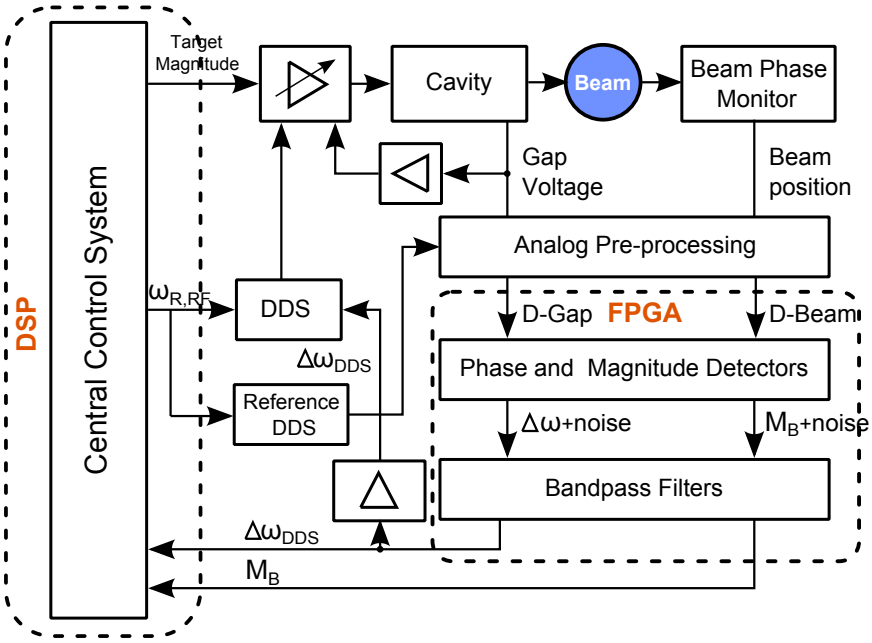


Fig. 5.1: Block diagram of the closed-loop control system for heavy ion synchrotron.

5.1 System Background

In antiproton and ion research, several synchrotrons and storage rings are used for many differential nuclear physic and material sciences. These machines in general deal with longitudinal oscillations of the particle bunches. Longitudinal bunch oscillation can be visualized by beam signals, where the beam current is injected in versus time. For bunched beams, the beam signal consists of a sequence of pulses due to the individual particle bunches. In normal cases, the pulses will generate a constant width beam signal. Acceleration process leads to beam signals with decreasing time periods of the pulses. The harmonic number h describes the number of bunches circulating in the synchrotron. Thus, every h^{th} pulse in the beam signal corresponds to the same particle bunch. Different modes of coherent longitudinal beam oscillations may occur due to an initial mismatch or to intensively dependent effect. These oscillations are characterized by their mode number m and n [60] and take place at the characteristic synchrotron frequency, which depends on the system state (more precisely, on the magnitude flux derivative, accelerating voltage, and particle energy). In order to eliminate undesired dipole oscillations, a beam phase control system has been devised, which was initially designed to deal with in-phase dipole oscillation (m=*1*, n=*0*) only [61]. The addition of amplitude detectors is intended to make it suitable for damping higher-order modes [60] [67].

The closed-loop control system for damping coherent dipole modes of oscillation is implemented by digital components like field programmable gate arrays (*FPGAs*) and digital signal processors (*DSPs*), due to flexibility, modularity and scalability for changing the control-loop parameters. The block diagram of the *DSPs* and *FPGAs* combination is presented in Fig. 5.1. The DDS unit generates an RF signal with angular frequency $\omega_{R,RF}+$

$\Delta\omega_{DDS}$, where $\omega_{R,RF} = \frac{2\pi}{T_R}$ denotes the angular revolution frequency of the reference particle. The signal *"Frequency Correction"* in the block diagram is a digital data stream and refers to $\Delta\omega_{DDS}$. The DDS drives the cavity through an amplifier chain producing a cavity RF signal that is called *"Gap voltage"* because the electric field is generated in a ceramic gap interrupting the metallic beam pipe.

5.2 Beam Phase Control

The diagram in Fig. 5.2 presents the signal-processing pipeline implemented in the heterogeneous system. It performs the following steps:

1. Some analog preprocessing is performed on both the *"Gap voltage"* and the beam position signal [59]. Note that the beam position signal is quite noisy and has a high level of uncertainly since the bunch may have an arbitrary shape.

2. Four successive samples are used to form the in-phase and quadrature components of both signals.

3. The phase of both signals is computed using Algorithm 25.

4. The phase difference is fed into a programmable FIR filter [140] in order to eliminate noise and other disturbances in the phase difference. The filter is tuned to a frequency slightly above [61] the synchrotron frequency, and has to be re-tuned continuously since this frequency changes.

5. The output of the filter is fed into a variable-gain controller, whose gain also depends on the characteristic frequency. This controller yields a frequency correction, which is then applied to the cavity in order to intentionally mis-tune it, thereby pulling the bunch back to its desired position.

BPM is the signal from the beam position monitor. u_{gap} is the voltage across the gap of the reference cavity. f_{ref} is the reference frequency [59]. Future extensions [67] are dashed. In order to enable real-time processing, the phase detector and filtering blocks have been implemented on an *FPGA*.

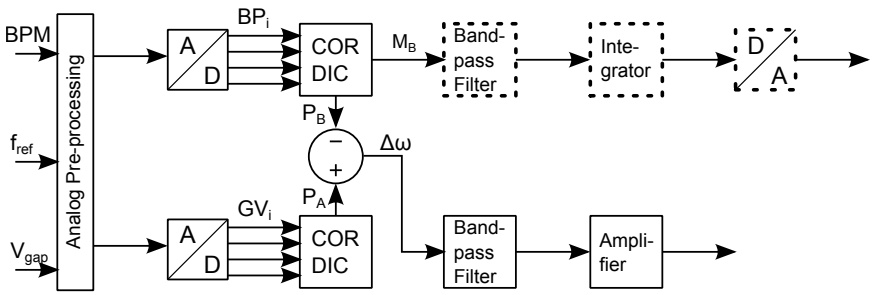


Fig. 5.2: Beam input reference and beam phase signals.

5.3 Phase-Magnitude Computation

5.3.1 State-of-the-Art

There are several methods to compute phase and magnitude. The look-up table *(LUT)* method is one of them, where the phase and magnitude value in every possible input is recorded. This method is simple and provides the lowest latency, but it has a drawback in that the size of memories required to store the phase and magnitude outputs is tremendous if the input bit-width is not small. A famous method is the CORDIC method, where the method calculates the phase and magnitude based on the rotation of a set of elementary angles. The CORDIC approach can be implemented without employing multipliers, but the computational latency is proportional to output precision requirements [75].

In this chapter, the proposed CORDIC methods, i.e. the double-rotation and the triple-rotation, are applied to compute the phase and magnitude difference of beam signals for a closed-loop control system. The proposed methods can alleviate the problems dealing with computational latency and accuracy which was described and analyzed in chapter 3. In addition, the convergence range of the proposed methods is extended by the sequential index extension method in order to improve the CORDIC's performance and efficiency.

5.3.2 Architecture for Phase-Magnitude Computing

The CORDIC algorithm in vectoring mode on the circular coordinate system provides the phase and magnitude computation as expressed in Equation 5.1

$$\begin{bmatrix} x_{out} \\ y_{out} \\ Z_{out} \end{bmatrix} = \begin{bmatrix} \sqrt{x_{in}^2 + y_{in}^2} \\ 0 \\ \arctan \dfrac{y_{in}}{x_{in}} \end{bmatrix} \tag{5.1}$$

The algorithms that are applied to perform the phase and magnitude computation based on the conventional, double-rotation, triple-rotation CORDIC methods with the sequential index extension method are illustrated in Algorithms 22, 23, and 24. The

functions *MICRO-CVCORDIC-VM*, *MICRO-DRCORDIC-VM*, and *MICRO-TRCORDIC-VM* in the algorithms conform to the CORDICs' micro-rotation in Equations 3.6, 3.9, and 3.12. To extend the convergence range, the sequential index i from $[-1, 0, \cdots, N-2]$ and $[-2, 0, \cdots, N-3]$ is determined for the double-rotation and triple-rotation methods, where N is the maximum number of iterations and $LTAN_{cv}$, $LTAN_{dr}$, and $LTAN_{tr}$ are the list of angles corresponding to 2^{-i}. The constant scaling factors (K_{cv}, K_{dr}, K_{tr}) for the conventional, double-rotation, triple-rotation methods are at 1.646768, 1.084716, and 1.007861, respectively.

Algorithm 22 CVCORDIC-VM

Input: : X_{in}, Y_{in}, N, $LTAN_{cv}$, K_{cv}^{-1}
Output: : X_{out}, Z_{out}
1: X_0=X_{in}, Y_0=Y_{in}, Z_{in}=0, δ_0=1 {Initialization}
2: **for** $i = 0\ to\ N-1$ **do**
3: **if** $Y_0 \geq 0$ **then**
4: δ_i=1;
5: **else**
6: δ_i=-1;
7: **end if**
8: [X, Y, Z]=MICRO-CVCORDIC-VM(X_o, Y_o, Z_o, $TLUT_{cv}(i)$, δ_i)
9: $X_0 = X$
10: $Y_0 = Y$
11: $Z_0 = Z$
12: **end for**
13: $X_{out} = X \cdot K_{cv}^{-1}$; Z_{out}=Z

Algorithm 25 shows the computation of the beam phase and magnitude based on online measurements of the gain voltages GV_i and beam position BP_i signals [42], which is illustrated in Fig. 5.2. The beam phase difference signal ($\Delta Phase$) is obtained from the difference between the phase detection signals P_A and P_B of the *"Gap voltages"* GV_i and *"Beam positions"* BP_i signals, respectively. After the iterative computation, X_{out} will be subtracted by 2.5 and 0.5666 for the double-rotation and triple-rotation methods.

The convergence range of the proposed CORDIC methods to perform the phase and magnitude computation is illustrated in Fig 5.3, where they are compared with Matlab/Simulation and the conventional method. With the sequential index extension method, the proposed methods can operate at $[0 : 0.9975]$ for any inputs x_{in} and y_{in}.

Algorithm 23 DRCORDIC-VM

Input: : X_{in}, Y_{in}, N, $LTAN_{dr}$, K_{dr}^{-1}
Output: : X_{out}, Z_{out}

X_0=X_{in}, Y_0=Y_{in}, Z_{in}=0, δ_0=1 {Initialization}
for $i = -1\ to\ N-2$ **do**
    if $Y_0 \geq 0$ **then**
        δ_i=1;
    else
        δ_i=-1;
    end if
    [X, Y, Z]=MICRO-DRCORDIC-VM(X_o, Y_o, Z_o, $TLUT_{dr}(i)$, δ_i)
    $X_0 = X$
    $Y_0 = Y$
    $Z_0 = Z$
end for
$X_{out} = (X \cdot K_{dr}^{-1}) - 1.5$; Z_{out}=Z

Algorithm 24 TRCORDIC-VM

Input: : X_{in}, Y_{in}, N, $LTAN_{tr}$, K_{tr}^{-1}
Output: : X_{out}, Z_{out}

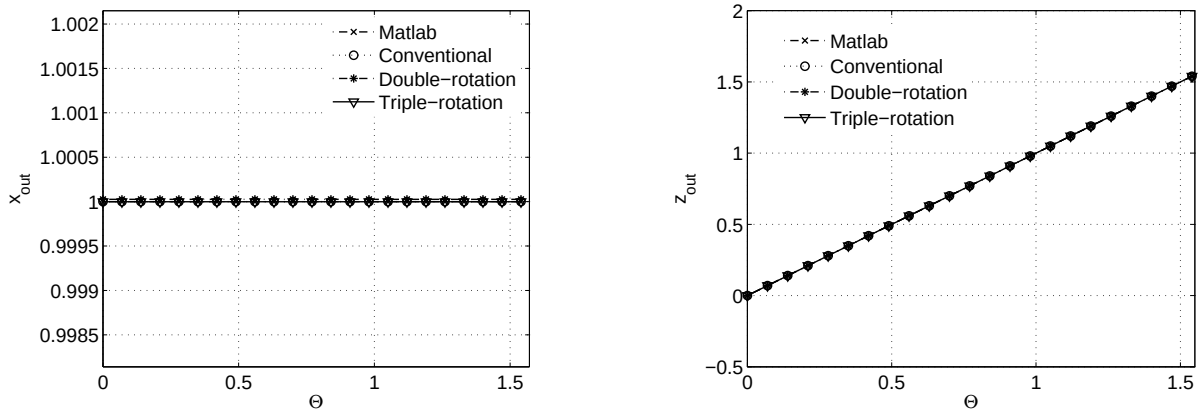
X_0=X_{in}, Y_0=Y_{in}, Z_{in}=0, δ_0=1 {Initialization}
for $i = -2\ to\ N-3$ **do**
    if $Y_0 \geq 0$ **then**
        δ_i=1;
    else
        δ_i=-1;
    end if
    [X, Y, Z]=MICRO-TRCORDIC-VM(X_o, Y_o, Z_o, $TLUT_{tr}(i)$, δ_i)
    $X_0 = X$
    $Y_0 = Y$
    $Z_0 = Z$
end for
$X_{out} = (X \cdot K_{tr}^{-1}) - 0.5666$; Z_{out}=Z

Algorithm 25 BEAM phase and magnitude

Input: : GV_1,GV_2,GV_3,GV_4,BP_1,BP_2,BP_3,BP_4

Output: : M, $\Delta\Phi$

1: Q_{1A} = GV_1; I_{1A} = GV_2; Q_{2A} = GV_3; I_{2A} = GV_4
2: Q_{1B} = BP_1; I_{1B} = BP_2; Q_{2B} = BP_3; I_{2B} = BP_4
3: ΔX_A = $Q_{1A} - Q_{2A}$
4: ΔY_A = $I_{1A} - I_{2A}$
5: {****** CORDIC which is in either convention, ******}
6: {****** double-rotation, or triple-rotation ******}
7: [M_B, P_B]=DRCORDIC-VM(ΔX_B, ΔY_B, N, $TLUT$,K_{dr}^{-1})
8: ΔX_B = $Q_{1B} - Q_{2B}$
9: ΔY_B = $I_{1B} - I_{2B}$
10: {****** CORDIC which is in either convention, ******}
11: {****** double-rotation, or triple-rotation ******}
12: [M_B, P_B]=DRCORDIC-VM(ΔX_B, ΔY_B, N, $TLUT$,K_{dr}^{-1})
13: M = M_B
14: $\Delta\Phi$=$P_A - P_B$



(a) $x_{out} = \sqrt{x_{in}^2 + y_{in}^2}$ (b) $z_{out} = tan^{-1}(\frac{y_{in}}{x_{in}})$

Fig. 5.3: Convergence range of the conventional, double-rotation and triple-rotation CORDIC methods to perform the phase and magnitude computation.

5.3.3 Verification and Simulation

Two input patterns of the *"Gap voltage"* and the *"Beam position"* of the system as shown in Fig. 5.1 are employed for the verification. The *"Gap voltage"* and the *"Beam position"* of each pattern will be the input of the beam phase and magnitude module, which is modeled by *VHDL* in 16-bit fixed-point format according to Algorithm 25. Since the module computes based on the CORDIC, the hardware simulation results will be obtained from the CORDIC in conventional, double-rotation, and triple-rotation methods. In order to

compare the computational accuracy in various fixed-point formats, different QI and QF is applied, i.e. $QI = 3$, $QF = 12$, $QI = 4$, $QF = 11$, $QI = 5$,and $QF = 10$. The input sampling rate of the phase detector module is at 0.8325 μsec and at 3.33 μsec for the bandpass filter module.

5.3.3.1 Test Pattern 1

In this pattern, the *"Gap voltage"* and *"Beam position"* signals are generated with the assumption that the magnitude of the *"Gap voltage"* signal is illustrated in Fig. 5.4. The phase and magnitude of the *"Beam position"* signal is changed when the beam accelerator is started.

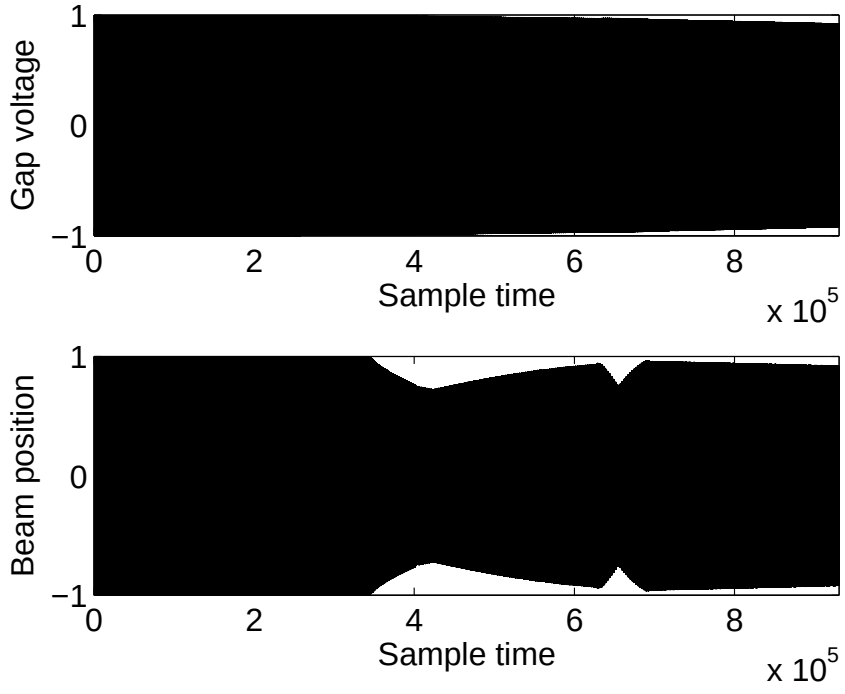


Fig. 5.4: Test pattern 1: *"Gap voltage"* and *"Beam position"* in digital signal for the phase detector module.

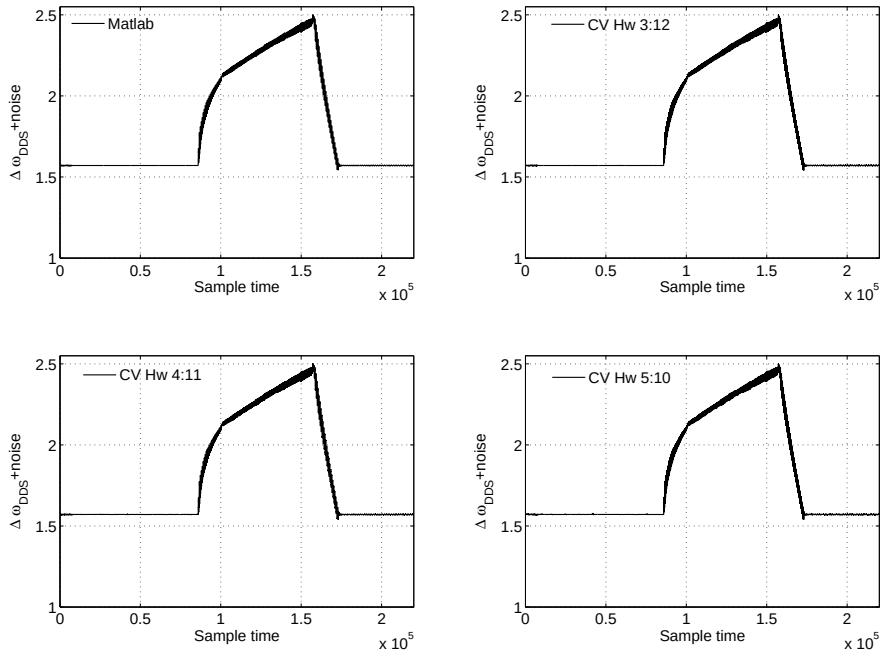


Fig. 5.5: Different phase ($\Delta\omega$) computational result of the conventional CORDIC method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

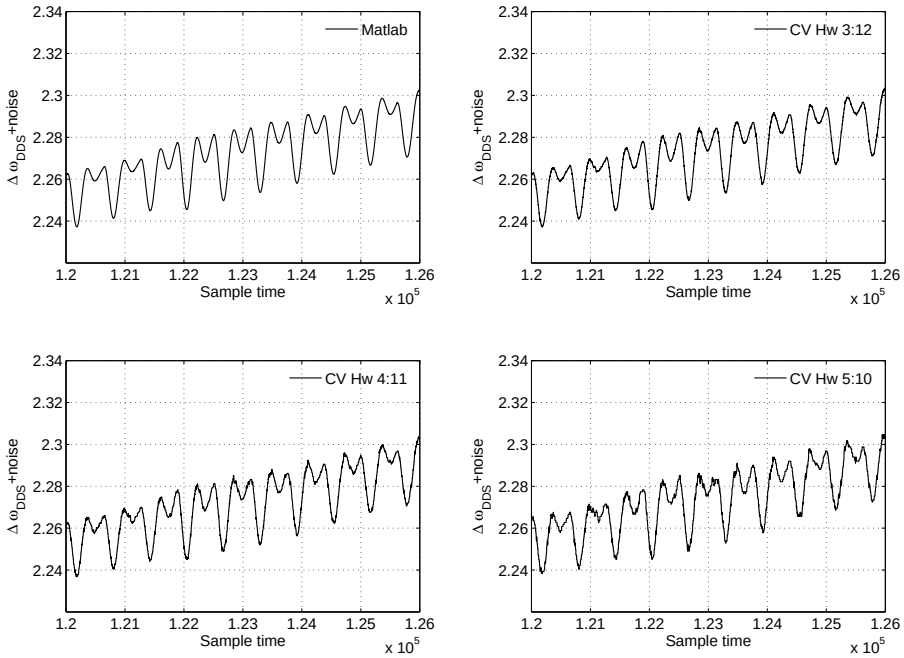


Fig. 5.6: Zoom of different phase ($\Delta\omega$) computational result of the conventional method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

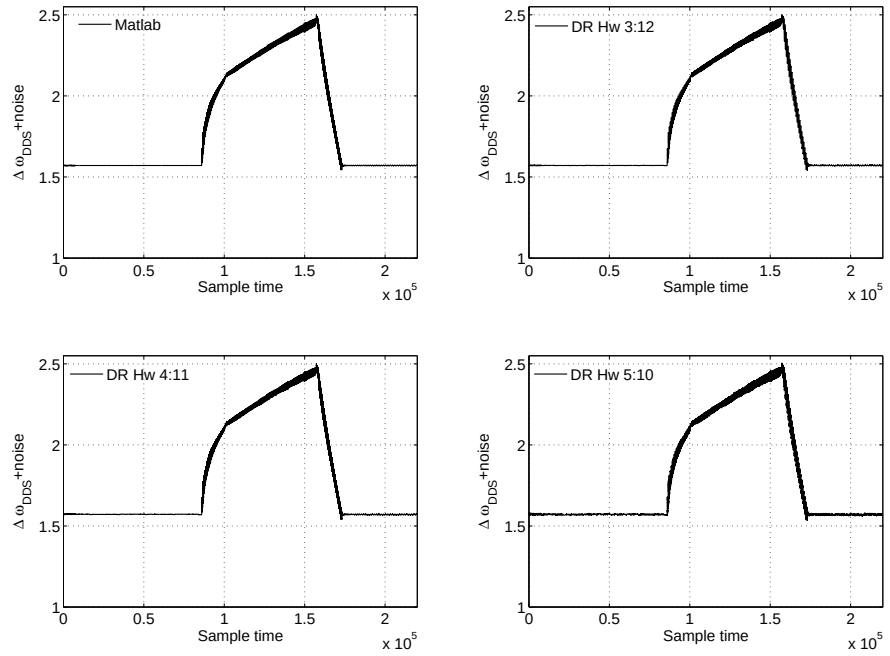


Fig. 5.7: Different phase ($\Delta\omega$) computational result of the double-rotation method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

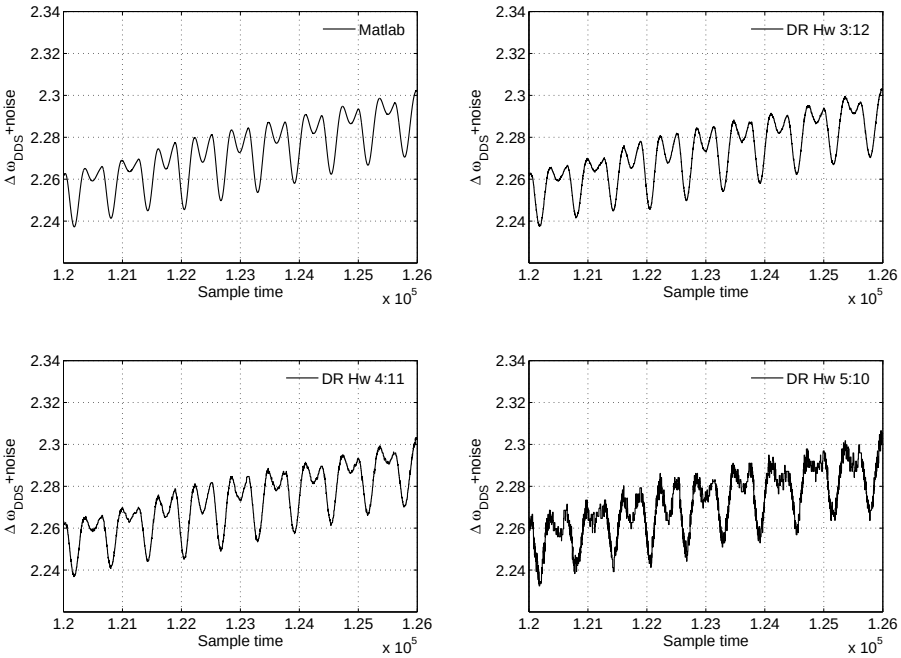


Fig. 5.8: Zoom of different phase ($\Delta\omega$) computational result of the double-rotation CORDIC method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

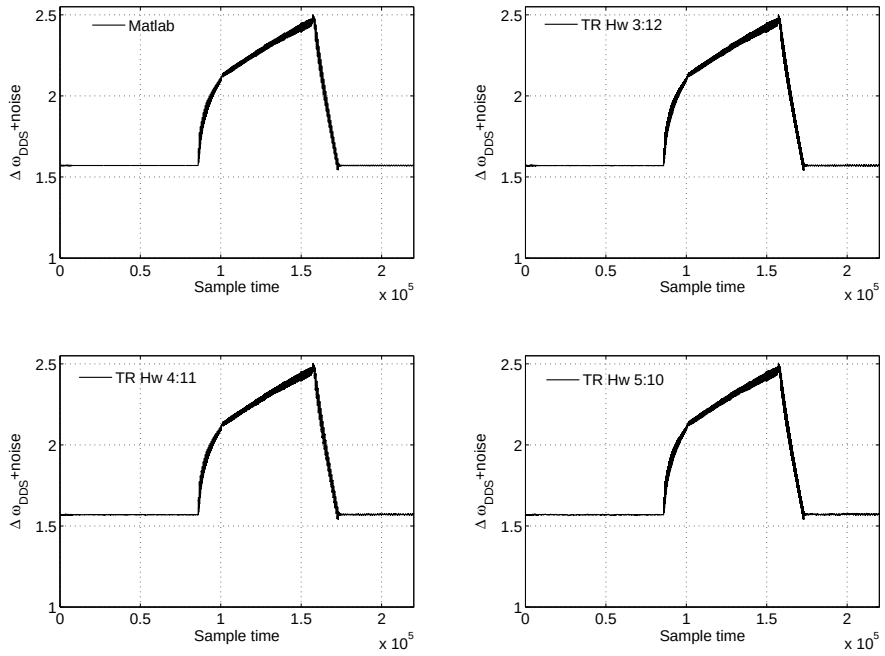


Fig. 5.9: Different phase ($\Delta\omega$) computational result of the triple-rotation CORDIC method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

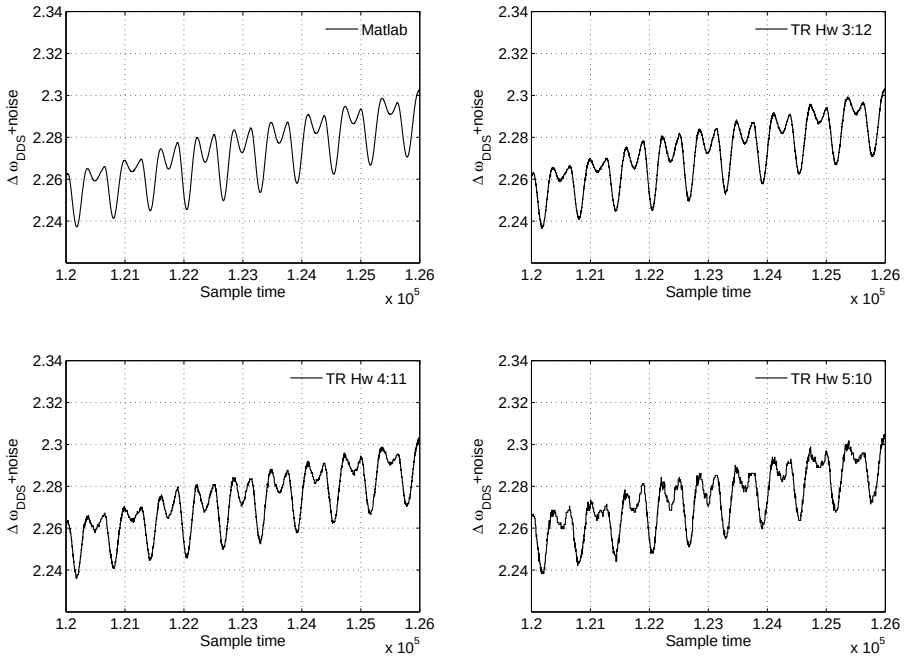


Fig. 5.10: Zoom of different phase ($\Delta\omega$) computational result of the triple-rotation CORDIC method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

Figs. 5.5, 5.7, and 5.9 present the hardware simulation results of the beam phase and magnitude module in the conventional, double-rotation, and triple-rotation methods. The simulation results with different fixed-point formats are compared with Matlab/Simulink results. The figures show that the hardware and software computations perform as expected. To visualize the precision in different fixed-point formats, the scaled up figures on each CORDIC method are illustrated in Figs 5.6, 5.8, and 5.10.

5.3.3.2 Test Pattern 2

The *"Gap voltage"* and *"Beam position"* signals in this pattern are generated from the actual ion synchrotron beam system. The phase and magnitude of the *"Gap voltage"* signal are fluctuated as shown in Fig. 5.11. The phase and magnitude of the *"Beam position"* signal is also changed. Notice that when the beam accelerator is started, the magnitude of the beam signal is at 0.5 approximately.

Figs. 5.12 to 5.17 show the hardware simulation results generated from the beam phase and magnitude module in different fixed-point formats.

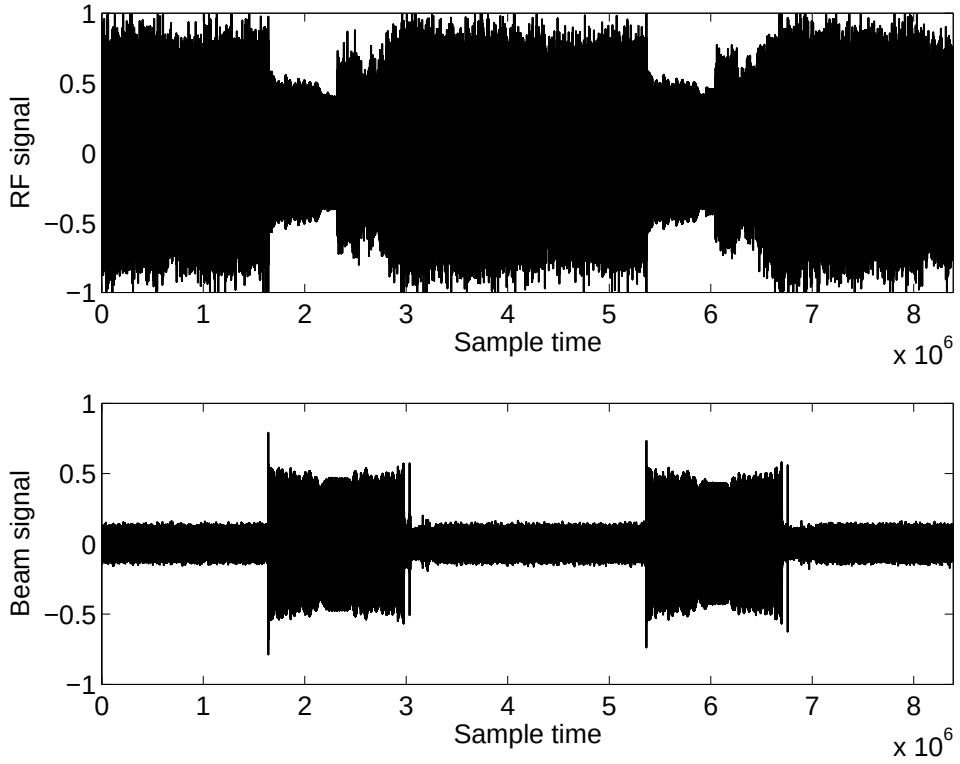


Fig. 5.11: Test pattern 2: *"Gap voltage"* and *"Beam position"* in digital signal for the phase detector module.

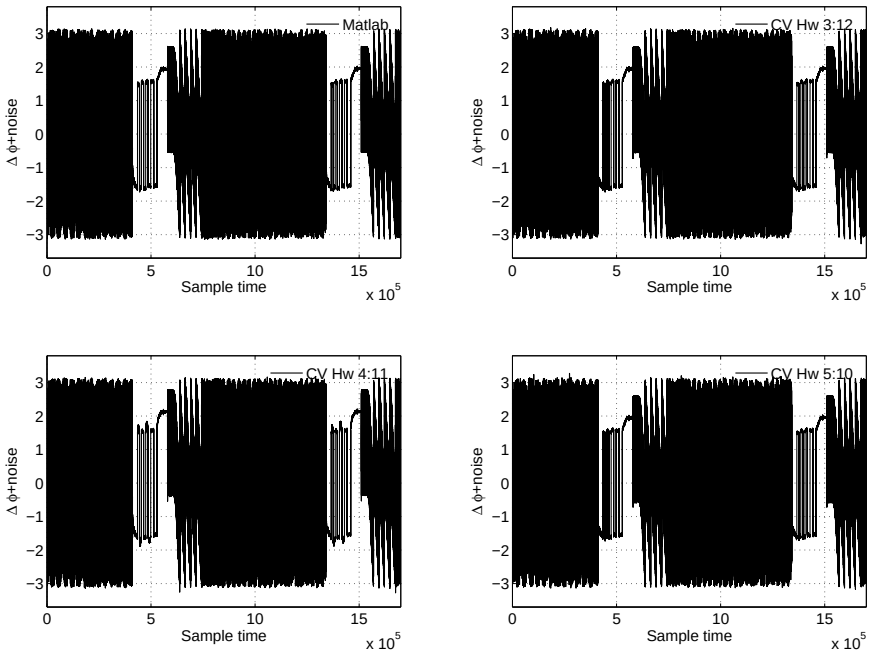


Fig. 5.12: Different phase ($\Delta\omega$) computational result of the conventional method on hardware fixed-point format $QI = 3$ $QF = 12$, $QI = 4$ $QF = 11$ and $QI = 5$ $QF = 10$

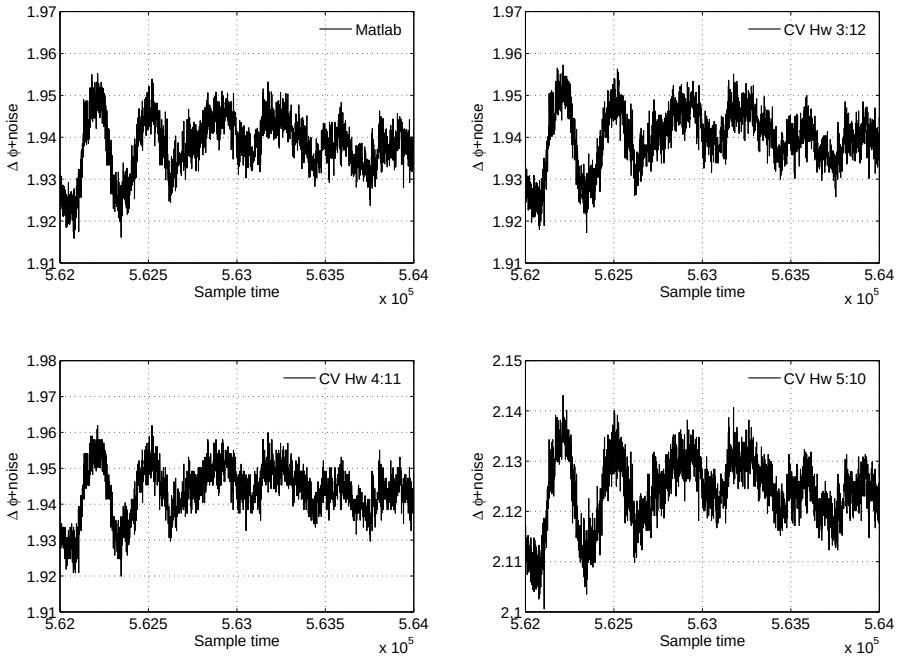


Fig. 5.13: Zoom of different phase ($\Delta\omega$) computational result of the conventional method on hardware fixed-point format $QI = 3$ $QF = 12$, $QI = 4$ $QF = 11$ and $QI = 5$ $QF = 10$

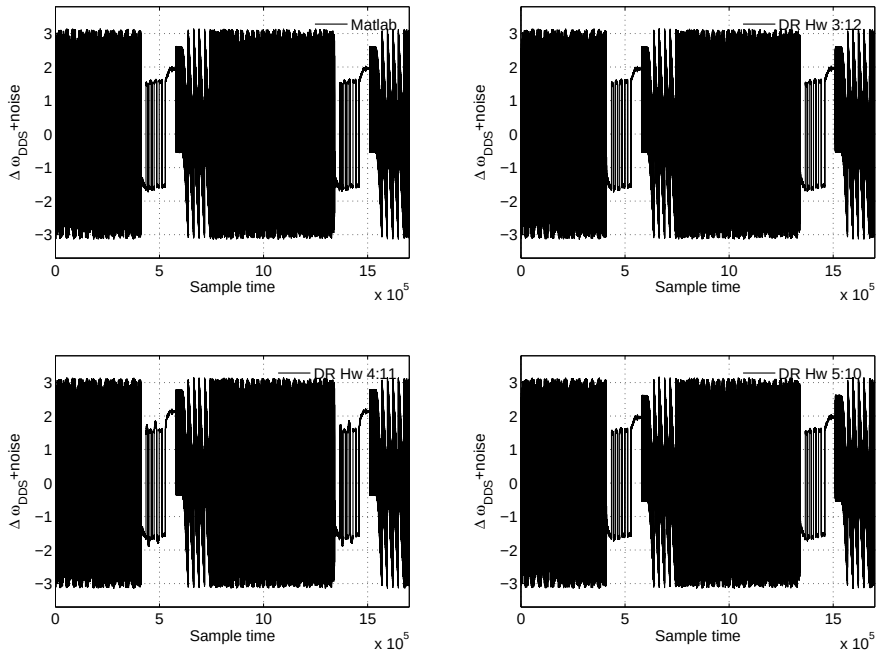


Fig. 5.14: Different phase ($\Delta\omega$) computational result of the double-rotation method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

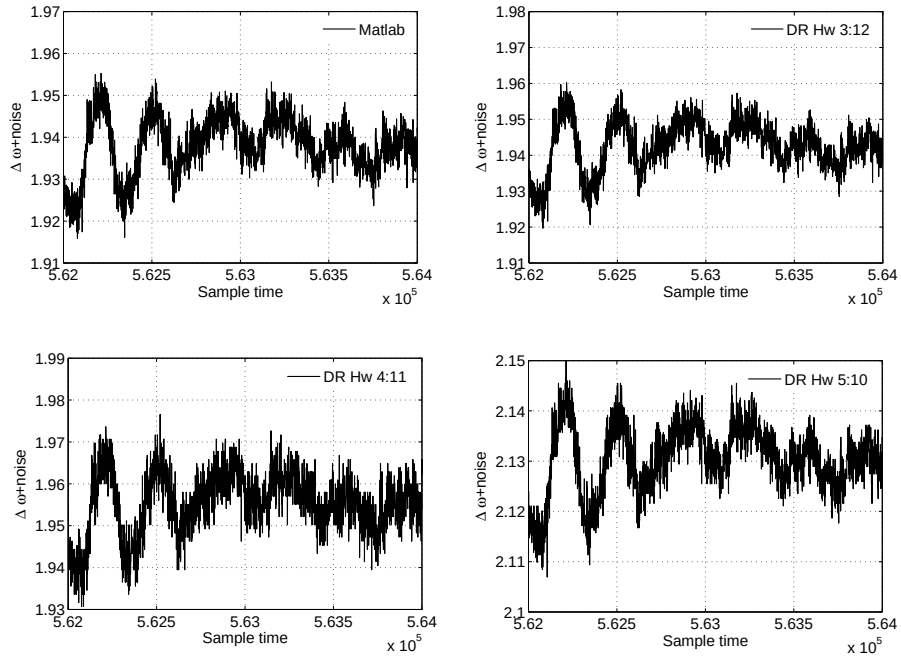


Fig. 5.15: Zoom of different phase ($\Delta\omega$) computational result of the double-rotation method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

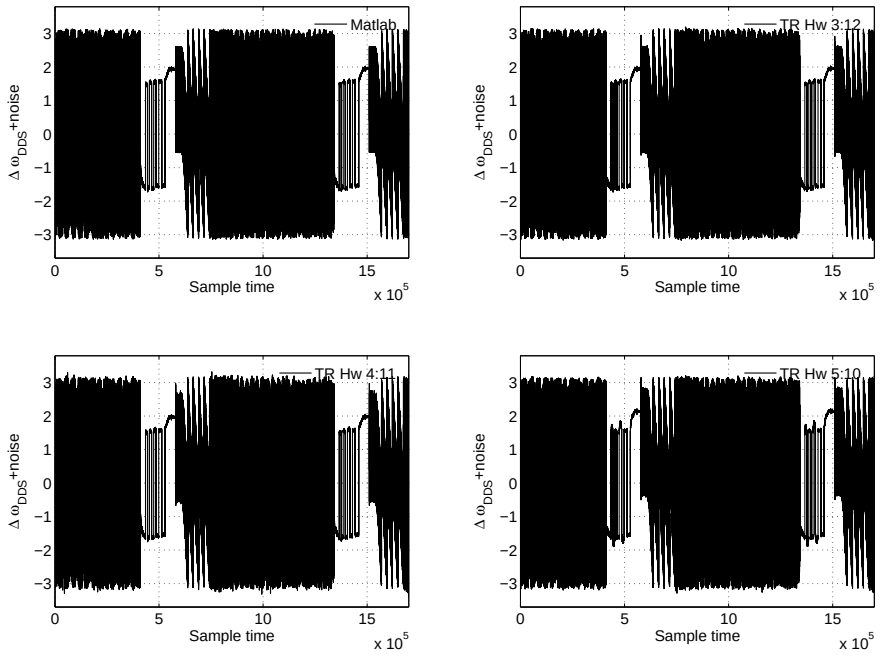


Fig. 5.16: Different phase ($\Delta\omega$) computational result of the triple-rotation method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

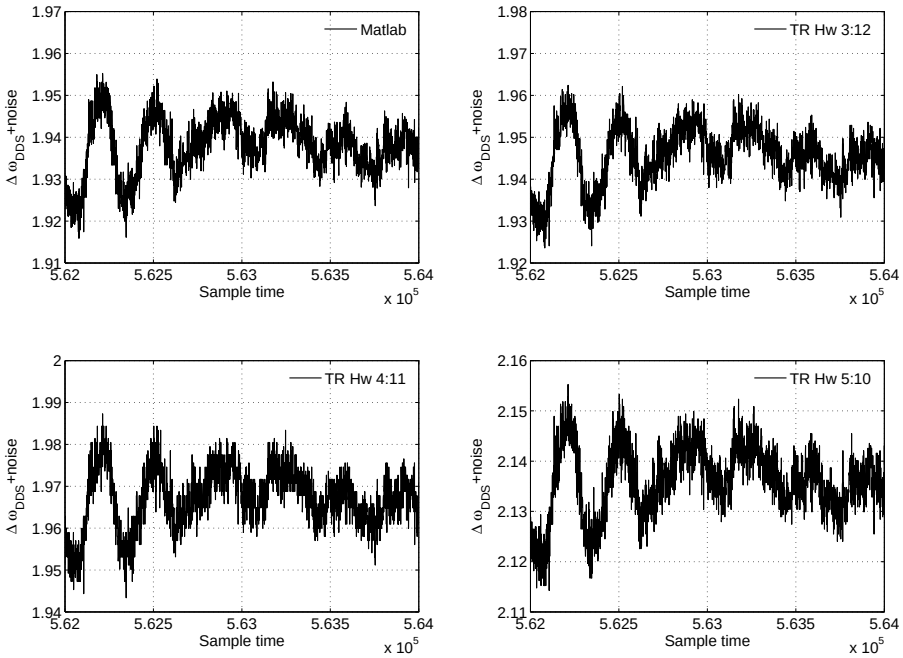


Fig. 5.17: Zoom of different phase ($\Delta\omega$) computational result of the triple-rotation method on hardware fixed-point format $QI = 3\ QF = 12$, $QI = 4\ QF = 11$ and $QI = 5\ QF = 10$

Tab. 5.1 shows the computational accuracy of the phase difference ($\Delta\omega$), which is computed by the conventional, double-rotation, and triple-rotation CORDIC methods. With the four statical analysis indicators, which are applied to measure the computational error of the $\Delta\omega$, the proposed methods provide better accuracy than the conventional method.

Tab. 5.1: The analysis of computational accuracy of the phase difference based on CORDIC methods.

	CORDIC	Statistical Analysis			
		$Max.\|error\|$	$Min.\|error\|$	$Ave.\|error\|$	$Std.Dev.\|error\|$
x_{out}	Conventional	4.6182E-7	2.9343E-7	1.5359E-7	1.468E-11
	Double-rotation	2.6258E-5	2.6257E-5	2.6257E-5	7.1442E-11
	Triple-rotation	1.0299E-6	1.0298E-6	1.0298E-6	3.0182E-11
z_{out}	Conventional	3.0392E-5	7.6653E-7	1.5207E-5	8.9097E-6
	Double-rotation	1.5178E-5	7.2580E-6	1.1698E-5	2.4857E-6
	Triple-rotation	1.0900E-5	3.3059E-7	5.3770E-6	3.3885E-6

5.4 Summary

The phase detection of the closed-loop control system for the heavy ion synchrotron application is applied to verify the proposed CORDIC methods, i.e. double-rotation and triple-rotation. The beam phase and magnitude algorithm based on the CORDIC is introduced. With the sequential index extension method, the proposed methods can support the convergence range of the input variable x and y from 0 to 0.9975. The two test patterns are applied, where the first is generated from the ideal magnitude (the constant magnitude), while the second is captured from the actual beam system. The hardware simulation results show that the beam phase and magnitude module can perform the phase difference efficiently.

Chapter 6

Concluding Remarks

Contents

The arithmetic algorithm, design, and architecture for *VLSI* implementation of the floating-point and fixed-point arithmetic unit have been presented and considered so far in the previous chapters. The standard and non-standard floating-point operators, i.e. adder/subtractor, multiplier, product-of-sum, and sum-of-product, as well as the elementary functions, i.e. sin, cos, sinh, cosh, rectan-to-polar etc., performed by the high precision CORDIC core are analysed and investigated in order to improve the performance and time efficiency. This chapter will summarize the contributions of the work (section 6.1) and some aspects related to potential future research investigations in the area of arithmetic unit research field (section 6.2).

6.1 Contribution of the Work

The contributions of this thesis are summarised as the following points.

1 *Improvement of the standard and non-standard floating-point operators with efficient method for hardware implementation in practice* [138], [142]: The floating-point arithmetic operators, i.e. adder/subtractor, multiplier, product-of-sum, and sum-of-product, are analysed and investigated in algorithmic forms in order to improve their performance. The right/left shifting function, the leading-one-detection function, and the integer multiplier, which are common functions, are proposed. The floating-point operators are considered to enhance the performance by the multiplexer-based shifting method, the binary-tree searching method, and the linear partial method, respectively.

2 *Design and analysis of extension-rotation CORDIC algorithms based on non-redundant method*, [144]: The CORDIC algorithm in the double-rotation and triple-rotation methods are proposed and investigated. The two methods accelerate the convergence by duplicating and triplicating the micro-rotation angles 2θ and 3θ. The non-redundant method is applied to obtain a constant scaling factor. The unified micro-rotation of the double-rotation and triple-rotation methods based on radix-2 available on rotation and vectoring modes in the circular, hyperbolic, and linear coordinate systems are introduced. The two measuring methods are presented for accuracy verification and comparison, i.e. *Mean Absolute Percent Error (MAPE)* and a set of statistical indicators consisting of the absolute maximum error, the absolute minimum error, the absolute mean error, and the absolute standard deviation error. These measuring methods are applied to verify and compare the computational accuracy and computational latency of the proposed CORDIC methods.

3 *Design, evaluation, and implementation of the fast CORDIC core and the floating-point arithmetic accelerator/processor* [143], [139], [137] : The proposed unified micro-rotations are designed and analysed based on the pipeline architecture. The floating-point arithmetic accelerator and the reconfigurable streaming processor are introduced for the case study. The objective of the accelerator is not only to improve the computational performance and efficiency of a classical main processor, but also to reduce the design cost. The reconfigurable streaming processor is designed to support the streaming data computation.

6.2 Direction for Future Work

In accordance with the current status of the research results presented in this thesis, future investigations to improve and extend the convergence range of the elementary functions performed by the double-rotation and triple-rotation CORDIC methods will remain open. The double-rotation and triple-rotation methods are introduced for the objective of reducing the latency and increasing the accuracy of the CORDIC computation. However, the reduction of the latency affects the convergence range of the proposed methods, which is smaller than the convergence range of the conventional method. Therefore, the extension of the convergence range of the elementary functions based on the double-rotation and triple-rotation methods in rotation and vectoring modes on the circular, hyperbolic, and linear coordinate systems will be a challenge for future work.

Appendix A

Hardware for Scientific and Engineering Applications

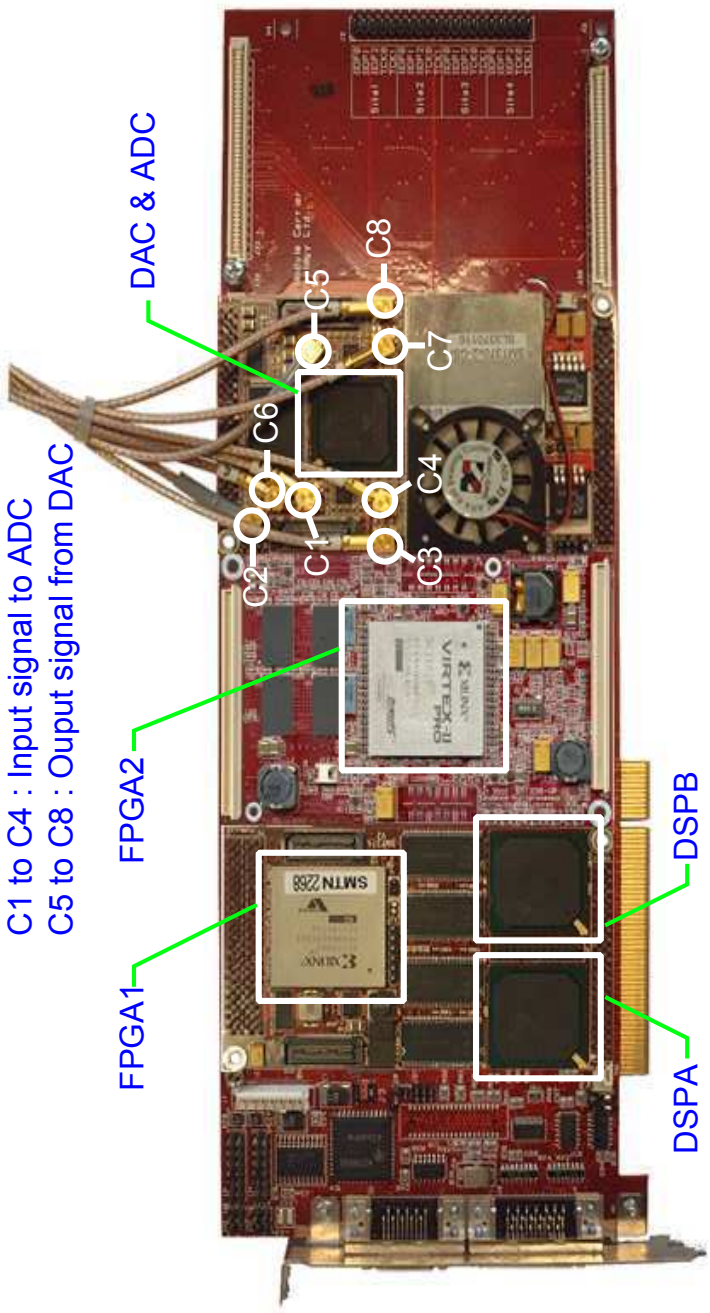


Fig. A.1: Heterogeneous platform for verification of the closed-loop control system in heavy ion synchrotron application.

Appendix B

Elementary rotation angle of the double-rotation and triple-rotation CORDIC methods

Tab. B.1: Elementary rotation angle of the double-rotation CORDIC method on the circular coordinate system

i	$2 \cdot tan^{-1}(2^{-i-1})$	i	$2 \cdot tan^{-1}(2^{-i-1})$
1	0.48996	33	1.1642E-10
2	0.24871	34	5.8208E-11
3	0.12484	35	2.9104E-11
4	6.2480E-2	36	1.4552E-11
5	3.1247E-2	37	7.276E-12
6	1.5625E-2	38	3.638E-12
7	7.8125E-3	39	1.819E-12
8	3.9062E-3	40	9.0949E-13
9	1.9531E-3	41	4.5475E-13
10	9.7656E-4	42	2.2737E-13
11	4.8828E-4	43	1.1369E-13
12	2.4414E-4	44	5.6843E-14
13	1.2207E-4	45	2.8422E-14
14	6.1035E-5	46	1.4211E-14
15	3.0518E-5	47	7.1054E-15
16	1.5259E-5	48	3.5527E-15
17	7.6294E-6	49	1.7764E-15
18	3.8147E-6	50	8.8818E-16
19	1.9073E-6	51	4.4409E-16
20	9.5367E-7	52	2.2204E-16
21	4.7684E-7	53	1.1102E-16
22	2.3842E-7	54	5.5511E-17
23	1.1921E-7	55	2.7756E-17
24	5.9605E-8	56	1.3878E-17
25	2.9802E-8	57	6.9389E-18
26	1.4901E-8	58	3.4694E-18
27	7.4506E-9	59	1.7347E-18
28	3.7253E-9	60	8.6736E-19
29	1.8626E-9	61	4.3368E-19
30	9.3132E-10	62	2.1684E-19
31	4.6566E-10	63	1.0842E-19
32	2.3283E-10	64	5.4210E-20

Tab. B.2: Elementary rotation angle of the double-rotation CORDIC method on the hyperbolic coordinate system

i	$2 \cdot tanh^{-1}(2^{-i-1})$	i	$2 \cdot tanh^{-1}(2^{-i-1})$
1	0.51083	33	1.1642E-10
2	0.25131	34	5.8208E-11
3	0.12516	35	2.9104E-11
4	6.252E-2	36	1.4552E-11
5	3.1253E-2	37	7.276E-12
6	1.5625E-2	38	3.638E-12
7	7.8125E-3	39	1.819E-12
8	3.9063E-3	40	9.0949E-13
9	1.9531E-3	41	4.5475E-13
10	9.7656E-4	42	2.2737E-13
11	4.8828E-4	43	1.1369E-13
12	2.4414E-4	44	5.6843E-14
13	1.2207E-4	45	2.8422E-14
14	6.1035E-5	46	1.4211E-14
15	3.0518E-5	47	7.1054E-15
16	1.5259E-5	48	3.5527E-15
17	7.6294E-6	49	1.7764E-15
18	3.8147E-6	50	8.8818E-16
19	1.9073E-6	51	4.4409E-16
20	9.5367E-7	52	2.2204E-16
21	4.7684E-7	53	1.1102E-16
22	2.3842E-7	54	5.5511E-17
23	1.1921E-7	55	2.7756E-17
24	5.9605E-8	56	1.3878E-17
25	2.9802E-8	57	6.9389E-18
26	1.4901E-8	58	3.4694E-18
27	7.4506E-9	59	1.7347E-18
28	3.7253E-9	60	8.6736E-19
29	1.8626E-9	61	4.3368E-19
30	9.3132E-10	62	2.1684E-19
31	4.6566E-10	63	1.0842E-19
32	2.3283E-10	64	5.4210E-20

Tab. B.3: Elementary rotation angle of the double-rotation CORDIC method on the linear coordinate system

i	$2 \cdot 2^{-i-1}$	i	$2 \cdot 2^{-i-1}$
1	0.5000	33	1.1642E-10
2	0.2500	34	5.8208E-11
3	0.1250	35	2.9104E-11
4	6.2500E-2	36	1.4552E-11
5	3.1250E-2	37	7.276E-12
6	1.5625E-2	38	3.638E-12
7	7.8125E-3	39	1.819E-12
8	3.9063E-3	40	9.0949E-13
9	1.9531E-3	41	4.5475E-13
10	9.7656E-4	42	2.2737E-13
11	4.8828E-4	43	1.1369E-13
12	2.4414E-4	44	5.6843E-14
13	1.2207E-4	45	2.8422E-14
14	6.1035E-5	46	1.4211E-14
15	3.0518E-5	47	7.1054E-15
16	1.5259E-5	48	3.5527E-15
17	7.6294E-6	49	1.7764E-15
18	3.8147E-6	50	8.8818E-16
19	1.9073E-6	51	4.4409E-16
20	9.5367E-7	52	2.2204E-16
21	4.7684E-7	53	1.1102E-16
22	2.3842E-7	54	5.5511E-17
23	1.1921E-7	55	2.7756E-17
24	5.9605E-8	56	1.3878E-17
25	2.9802E-8	57	6.9389E-18
26	1.4901E-8	58	3.4694E-18
27	7.4506E-9	59	1.7347E-18
28	3.7253E-9	60	8.6736E-19
29	1.8626E-9	61	4.3368E-19
30	9.3132E-10	62	2.1684E-19
31	4.6566E-10	63	1.0842E-19
32	2.3283E-10	64	5.4210E-20

Tab. B.4: Elementary rotation angle of the triple-rotation CORDIC method on the circular coordinate system

i	$3 \cdot tan^{-1}(2^{-i-2})$	i	$3 \cdot tan^{-1}(2^{-i-2})$
1	-	33	8.7311E-11
2	1.8726E-2	34	4.3656E-11
3	9.3720E-2	35	2.1828E-11
4	4.6871E-2	36	1.0914E-11
5	2.3437E-2	37	5.4570E-12
6	1.1719E-2	38	2.7285E-12
7	5.8594E-3	39	1.3642E-12
8	2.9297E-3	40	6.8212E-13
9	1.4648E-3	41	3.4106E-13
10	7.3242E-4	42	1.7053E-13
11	3.6621E-4	43	8.5265E-14
12	1.8311E-4	44	4.2633E-14
13	9.1553E-5	45	2.1316E-14
14	4.5776E-5	46	1.0658E-14
15	2.2888E-5	47	5.3291E-15
16	1.1444E-5	48	2.6645E-15
17	5.7220E-6	49	1.3323E-15
18	2.8610E-6	50	6.6613E-16
19	1.4305E-6	51	3.3307E-16
20	7.1526E-7	52	1.6653E-16
21	3.5763E-7	53	8.3267E-17
22	1.7881E-7	54	4.1633E-17
23	8.9407E-8	55	2.0817E-17
24	4.4703E-8	56	1.0408E-17
25	2.2352E-8	57	5.2042E-18
26	1.1176E-8	58	2.6021E-18
27	5.5879E-9	59	1.3010E-18
28	2.7940E-9	60	6.5052E-19
29	1.3970E-9	61	3.2526E-19
30	6.9849E-10	62	1.6263E-19
31	3.4925E-10	63	8.1315E-20
32	1.7462E-10	64	1.3553E-20

Tab. B.5: Elementary rotation angle of the triple-rotation CORDIC method on the hyperbolic coordinate system

i	$3 \cdot tanh^{-1}(2^{-i-2})$	i	$3 \cdot tanh^{-1}(2^{-i-2})$
1	-	33	8.7311E-11
2	0.18774	34	4.3656E-11
3	9.3781E-2	35	2.1828E-11
4	4.6879E-2	36	1.0914E-11
5	2.3438E-2	37	5.457E-12
6	1.1719E-2	38	2.7285E-12
7	5.8594E-3	39	1.3642E-12
8	2.9297E-3	40	6.8212E-13
9	1.4648E-3	41	3.4106E-13
10	7.3242E-4	42	1.7053E-13
11	3.6621E-4	43	8.5265E-14
12	1.8311E-4	44	4.2633E-14
13	9.1553E-5	45	2.1316E-14
14	4.5776E-5	46	1.0658E-14
15	2.2888E-5	47	5.3291E-15
16	1.1444E-5	48	2.6645E-15
17	5.7220E-6	49	1.3323E-15
18	2.8610E-6	50	6.6613E-16
19	1.4305E-6	51	3.3307E-16
20	7.1526E-7	52	1.6653E-16
21	3.5763E-7	53	8.3267E-17
22	1.7881E-7	54	4.1633E-17
23	8.9407E-8	55	2.0817E-17
24	4.4703E-8	56	1.0408E-17
25	2.2352E-8	57	5.2042E-18
26	1.1176E-8	58	2.6021E-18
27	5.5879E-9	59	1.3010E-18
28	2.794E-9	60	6.5052E-19
29	1.397E-9	61	3.2526E-19
30	6.9849E-10	62	1.6263E-19
31	3.4925E-10	63	8.1315E-20
32	1.7462E-10	64	4.0658E-20

Tab. B.6: Elementary rotation angle of the triple-rotation CORDIC method on the linear coordinate system

i	$3 \cdot 2^{-i-2}$	i	$3 \cdot 2^{-i-2}$
1	-	33	8.7311E-11
2	0.1875	34	4.3656E-11
3	9.375E-2	35	2.1828E-11
4	4.6875E-2	36	1.0914E-11
5	2.3438E-2	37	5.457E-12
6	1.1719E-2	38	2.7285E-12
7	5.8594E-3	39	1.3642E-12
8	2.9297E-3	40	6.8212E-13
9	1.4648E-3	41	3.4106E-13
10	7.3242E-4	42	1.7053E-13
11	3.6621E-4	43	8.5265E-14
12	1.8311E-4	44	4.2633E-14
13	9.1553E-5	45	2.1316E-14
14	4.5776E-5	46	1.0658E-14
15	2.2888E-5	47	5.3291E-15
16	1.1444E-5	48	2.6645E-15
17	5.7220E-6	49	1.3323E-15
18	2.8610E-6	50	6.6613E-16
19	1.4305E-6	51	3.3307E-16
20	7.1526E-7	52	1.6653E-16
21	3.5763E-7	53	8.3267E-17
22	1.7881E-7	54	4.1633E-17
23	8.9407E-8	55	2.0817E-17
24	4.4703E-8	56	1.0408E-17
25	2.2352E-8	57	5.2042E-18
26	1.1176E-8	58	2.6021E-18
27	5.5879E-9	59	1.3010E-18
28	2.794E-9	60	6.5052E-19
29	1.397E-9	61	3.2526E-19
30	6.9849E-10	62	1.6263E-19
31	3.4925E-10	63	8.1315E-20
32	1.7462E-10	64	4.0658E-20

References

[1] K. H. Abed and R. E. Siferd. "VLSI Implementations of Low-Power Leading-One Detector Circuits ". In *Proceedings of the IEEE SoutheastCon*, pages 279–284, 2006.

[2] F. Angarita, A. Perez-Pascual, T. Sansaloni, and J. Vails. "Efficient FPGA implementation of Cordic algorithm for circular and linear coordinates". *International Conference on Field Programmable Logic and Applications*, pages 535–538, 2005.

[3] E. Antelo, J. D. Bruguera, and E. L. Zapata. "Unified mixed radix 2-4 redundant CORDIC processor". *IEEE Transactions on Computers*, 45:1068–1073, 1996.

[4] E. Antelo, T. Lang, and J. D. Bruguera. "Very-high radix cicular CORDIC: Vecotring and unified rotation/vectoring". *IEEE Trans. Comput.*, 49(7):727–739, July 2000.

[5] E. Antelo, J. Villalaba, D. Bruguera, and E. Zapata. "High performance rotaion architecture based on the radix CORDIC algorithm". *IEEE Trans. Comput.*, 46(46):855–870, August 1997.

[6] T. Aoki, I. Kitaori, and T. Higuchi. "Radix-2-4-8 CORDIC for Fast Vector Rotation". *IEICE Transaction Fundamentals*, E83-A(6):1106–1114, 2000.

[7] M. Arnold and S. Collange. "A Real/Complex Logarithmic Number System ALU" . *IEEE Transactions on Computers*, 60:202–213, 2011.

[8] G. H. A. Aty, A. I. Hussein, I. S. Ashour, and M. Mones. "High-speed, area-efficient FPGA-based floating-point multiplier". In *Proceedings of the 15th International Conference on Microelectronics*, pages 274–277, 2003.

[9] D. Bader, J. Jaja, D. Harwood, and L. Davis. "Parallel algorithms for image enhancement and segmentation by region growing with an experimental study ". *The 10th International Parallel Processing Symposium*, pages 414–423, 1996.

[10] M. Baesler, S. Voigt, and T. Teufel. "An IEEE 754-2008 Decimal Parallel and Pipelined FPGA Floating-Point Multiplier ". In *International Conference on Field Programmable Logic and Applications*, pages 489–495, 2010.

[11] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. Lim. "Reduced latency IEEE floating-point standard adder architectures". In *Proceedings. 14th IEEE Symposium on Computer Arithmetic, 1999.*, pages 35–42, 1999.

[12] D. E. Bernholdt, W. R. Elwasif, and J. A. Kohl. "A Component Architecture for High-Performance Computing". *Workshop on Performance Optimization for High-Level Languages and Libraries, New York,*, pages 1–10, 2002.

[13] S. Biehl and D. Mayer. "Dynamic Characterisation of Piezo resistive Sensor Systems for Adaptronic Devices". In *IEEE International Symposium on Industrial Electronics*, 2007.

[14] L. Bleris, P. Vouzis, M. Arnold, and M. Kothare. "A co-processor FPGA platform for the implementation of real-time model predictive control". In *American Control Conference*, 2006.

[15] B. Bomar and B. Winkleman. "A method for accelerating the design of optimal linear-phase FIR digital filters". *IEEE Transactions on Signal Processing*, 39(6):1419–1421, June 1991.

[16] J. Bruguera and T. Lang. "Leading-one prediction with concurrent position correction". *IEEE Transactions on Computers*, 48(10):1083–1097, October 1999.

[17] J. R. Cavallaro and A. C. Elster. "A CORDIC Processor Array for SVD of a Complex Matrix". *Elsevier Science Algorithms, Analysis and Applications*, pages 227–239, 1991.

[18] S. Chai, S. Chiricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, and A. Lopez-Lagunas. "Streaming processors for next-generation mobile imaging applications ". *IEEE Communications Magazine*, 43:81–89, 2005.

[19] M. Chakraborty, S. Pervin, and T. Lamba. "A hyperbolic LMS algorithm for CORDIC based realization" . In *Proceedings of the 11th IEEE Signal Processing Workshop on Statistical Signal Processing*, pages 373–376, 2001.

[20] C. Chen, L.-A. Chen, and J.-R. Cheng. "Architectureal design of a fast floating-point multiplication-add fused unit using signed-digit addition". *IEE Proceedings - Computers and Digital Techniques*, 149(4):113–120, July 2002.

[21] D. Cochrab. "Algorithms and accuracy in the HP-35". *HewlettPackard Journal*, pages 1–11, June 1972.

[22] J. Cong and H. Yean-Yow. "Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping" . *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20:1077–1090, 2001.

[23] F. Corneliu, A. Felix, V. Constantin, and D. Alexandru. "Logarithmic tools for in-camera image processing ". *IET Irish Signals and Systems Conference, ISSC 2008*, pages 394–399, 2008.

[24] R. Cumplido, S. Jones, R. Goodall, and S. Bateman. "A high-performance processor for embedded real-time control ". *IEEE Transactions on Control Systems Technology*, 13:485–492, 2005.

[25] F. Dahlgren and J. Torrellas. "Cache-only memory architectures". *Computer*, 32:72–79, 1999.

[26] M. Darley, B. Kronlage, D. Bural, B. Churchill, D. Pulling, P. Wang, R. Iwamoto, and L. Yang. "The TMS390C602A floating-point coprocessor for Sparc systems". *IEEE Micro*, pages 36–47, 1990.

[27] F. de Dinechin, B. Pasca, O. Cret, and R. Tudoran. "An FPGA-specific approach to floating-point accumulation and sum-of-products". In *International Conference on ICECE Technology*, pages 33–40, 2008.

[28] J. Delosme. "VLSI implementation of rotation in pseudo Euclidean spaces". *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 927–930, 1983.

[29] K. Donghyun and K. Lee-Sup. "A Floating-Point Unit for 4D Vector Inner Product with Reduced Latency". *IEEE Transactions on Computers*, 58(7):890–901, July 2009.

[30] M. D. Ercegovac and T. Lang. "Redundant and on-line CORDIC: Application to matrix triangularization and SVD". *IEEE Trans. Comput.*, 39(6):725–740, June 1990.

[31] J. Euh, J. Chittamuru, and W. Burleson. "CORDIC vector interpolator for power-aware 3D computer graphics". In *IEEE Workshop on Signal Processing Systems*, 2002.

[32] S. F.Hsiao and J. M. Delosme. "Householder CORDIC algorithm". *IEEE Trans. Computers*, 44(8):990–1001, 1995.

[33] S. F.Hsiao and J. M. Delosme. "Parallel singular value decomposition of complex matrices using multi-dimensional CORDIC algorithms". *IEEE Trans. Signal Processing*, 44(3):685–697, 1996.

[34] M. Fitz and W. Lindsey. "Decision-directed burst-mode carrier synchronization techniques ". *IEEE Transactions on Communications*, 40:1644–1653, 1992.

[35] L. Ganghee, C. Kiyoung, and N. Dutt. "Mapping Multi-Domain Applications Onto Coarse-Grained Reconfigurable Architectures ". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30:637–650, 2011.

[36] H. Garraway. "Parallel Computer Architecture: A Hardware/Software Approach". *IEEE Concurrency*, 7:83–84, 1999.

[37] J. Gray, F. Grenzow, and M. Siedband. "Applying a PC accelerator board for medical imaging". *IEEE Engineering in Medicine and Biology Magazine*, 9:61–63, 1990.

[38] M. Grossard, C. Rotinat-Libersa, and N. Chaillet. "Redesign of the MMOC micro-gripper piezoactuator using a new topological optimization method ". In *International conference on Advanced intelligent mechatronics*, 2007.

[39] A. Grushin and M. Remizov. "Fast result normalization in FP adder". In *IEEE 25th Convention of Electrical and Electronics Engineers in Israel*, pages 152–156, 2008.

[40] C. Guangyu, L. Feihui, S. Son, and M. Kandemir. "Application mapping for chip multiprocessors" . In *Design Automation Conference*, number 620–625, 2008.

[41] C. Guannan, C. Rong, C. Jianxin, X. Zhiming, H. Zufang, F. Shangyuan, L. Yongzeng, and Y. Kuntao. "A Gray-Natural Logarithm Ratio Neighborhood Filters Method for Biomedical Image Denoising". *The 2nd International Conference on Bioinformatics and Biomedical Engineering*, pages 2605–2608, 2008.

[42] A. Guntoro, P. Zipf, O. Soffke, H. Klingbeil, M. Kumm, and M. Glesner. "Implementation of Realtime and Highspeed Phase Detector on FPGA". In *Proc. Int'l Workshop on Applied Reconfigurable Computing*, pages 1–11, 2006.

[43] W. Han, Z. Yousi, and L. Xiaokang. "A Parallel Double-Step CORDIC Algorithm for Digital Down Converter ". In *Communication Networks and Services Research Conference, 2009.*, 2009.

[44] R. Harber, X. Hu, J. Li, and S. Bass. "The application of bit-serial CORDIC computational units to the design of inverse kinematics processors". *IEEE International Conference on Robotics and Automation*, 2:1152–1157, 1988.

[45] R. Hashemian. "Memory efficient and high-speed search Huffman coding ". *IEEE Transactions on Communications*, 43:2576–2581, 1995.

[46] P. Hatcher, M. Quinn, A. Lapadula, B. Seevers, R. Anderson, and R. Jones. "Data-parallel programming on MIMD computers ". *IEEE Transactions on Parallel and Distributed Systems*, 2(3):377–383, 1991.

[47] S. Hitotumatu. "Complex arithmetic through CORDIC". *Kidau Math. Sem. Rep.*, 26:176–186, 1974.

[48] E. Hokenek and R. Montoye. "Leading-Zero Anticipator (LZA) in the IBM RISC System/6000 Floating-Point Execution Unit". *IBM Journal of Research and Development*, pages 71–77, 1990.

[49] S.-F. Hsiao and J.-Y. Chen. "Design, Implementation and Analysis of a New Redundant CORDIC Processor with Constant Scaling Factor and Regular Structure". *J. VLSI Signal Process. Syst.*, 20:267–278, December 1998.

[50] X. Hu, R. Harber, and S. Bass. "Expanding the range of convergence of the CORDIC algorithm ". *IEEE Transactions on Computers*, 40:13–21, 1991.

[51] X. Huang, M. Li, H. Hu, and H. Lu. "Geomagnetism GPS SINS Integrated Navigation Based on an Improved Square-Root UKF ". In *Second International Conference on Intelligent Computation Technology and Automation*, 2009.

[52] C. Huntsman and D. Cawthron. "The MC68881 Floating-point Coprocessor". *IEEE Micro*, 3:44–54, 1983.

[53] IEEE. *"IEEE Standard for Binary Floating-Point Arithmetic"*. Number ANSI/IEEE Std 754-1985. 1985.

[54] S. Jain, V. Erraguntla, S. Vangal, Y. Hoskote, N. Borkar, T. Mandepudi, and V. Karthik. "A 90mW/GFlop 3.4GHz Reconfigurable Fused/Continuous Multiply-Accumulator for Floating-Point and Integer Operands in 65nm". In *International Conference on VLSI Design*, pages 252–257, 2010.

[55] S. Ju-Ho, W. Jeong-Ho, J. Yoo, and Y. Hoi-Jun. "Design and Test of Fixed-point Multimedia Co-processor for Mobile Applications". In *Proceedings Design, Automation and Test in Europe*, pages 1–5, 2006.

[56] S. Kang, O. Changhwan, and D. Sung. "Performance evaluation of a high-speed ATM switch with multiple common memories ". *IEEE Transactions on Communications*, 50:332–340, 2002.

[57] P. Karlström, A. Ehliar, and D. Liu. "High-performance, low-latency field-programmable gate array-based floating-point adder and multiplier units in a Virtex 4". *Computers & Digital Techniques, IET*, 2:305–313, January 2008.

[58] S. Kawasaki, M. Watabe, and S. Morinaga. "A floating-point VLSI chip for the TRON architecture: an architecture for reliable numerical programming". *IEEE Micro*, 9:26–44, 1989.

[59] H. Klingbeil. "A Fast DSP-Based Phase-Detector for Closed-Loop RF Control in Synchrotrons". *IEEE Trans. Instrumentation and Measurement*, 54(3):1209–1213, June 2005.

[60] H. Klingbeil, D. Lens, M. Mehler, and B. Zipf. "Modeling Longitudinal Oscillations of Bunched Beams in Synchrotrons".

[61] H. Klingbeil, B. Zipfel, M. Kumm, and P. Moritz. "A Digital Beam-Phase Control System for Heavy-Ion Synchrotrons". *IEEE Trans. Nuclear Science*, 54(6):2604–2610, Dec. 2007.

[62] M. Kuhlmann and K. Parhi. "Fast low-power shared division and square-root architecture ". In *Proceedings. International Conference on Computer Design: VLSI in Computers and Processors*, pages 128–135, 1998.

[63] B. Lakshmi and A. S. Dhar. "CORDIC Architectures: A Survey". *Hindawi Publishing Corporation VLSI Design*, 2010:1–19, 2010.

[64] C. S. Lee and P. Chang. "A maximum pipelined CORDIC architecture for inverse kinematic position computation" . *IEEE Journal of Robotics and Automation*, 3:445–458, 1987.

[65] S.-W. Lee, K.-S. Kwon, and I.-C. Park. "Pipelined Cartesian-to-Polar Coordinate Conversion Based on SRT Division". *IEEE Transactions on Circuits and Systems II: Express Briefs*, 54:680–684, 2007.

[66] S. Leeke and L. Maharatna. "A low-power geometric mapping co-processor for high-speed graphics application ". In *Proceedings. 2006 IEEE International Symposium on Circuits and Systems*, 2006.

[67] D. Lens, H. Klingbeil, T. Guner, A. Popescu, and K. Groß. "Damping of Longitudinal Modes in Heavy-Ion Synchrotrons by RF-Feedback". In *Proc. IEEE Conf. on Control Applications*, pages 1737–1742, 2010.

[68] C.-C. Li and S.-G. Chen. "A radix-4 redundant CORDIC algorithm with fast on-line variable scale factor compensation". In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 639–642, 1997.

[69] W. Ligon, S. McMillan, G. Monn, F. Stivers, and K. Underwood. "A revaluation of the practicality of floating-point operations on FPGAs". In *IEEE Symp. FPGAs for Custom Computing Machines*, pages 206–215, April 1998.

[70] H. Makino, H. Suzuki, H. Morinaka, Y. Nakase, K. Mashiko, and T. Sumi. "A 286 MHz 64-b floating point multiplier with enhanced CG operation". *IEEE Journal of Solid-State Circuits*, 31:504–513, 1996.

[71] A. Malik, D. Chen, Y. Choi, M. H. Lee, and S.-B. Ko. "Design tradeoff analysis of floating-point adders in FPGAs". *Canadian Journal of Electrical and Computer Engineering*, 33:169–175, 2008.

[72] A. Malik and S.-B. Ko. "A Study on the Floating-Point Adder in FPGAS ". In *Canadian Conference on Electrical and Computer Engineering*, pages 86–89, 2006.

[73] P. Maurer. "Design verification of the WE 32106 math accelerator unit". *IEEE Design & Test of Computers*, 5:11–21, 1988.

[74] F. Mayer-Lindenberg and V. Beller. "An FPGA-based floating-point processor array supporting a high-precision dot product". In *IEEE International Conference on Field Programmable Technology*, pages 317–320, 2006.

[75] P. Meher, J. Valls, J. Tso-Bing, K. Sridharan, and K. Maharatna. "50 Years of CORDIC: Algorithms, Architectures, and Applications". *IEEE Transactions, Circuits and Systems*, 56:1893–1907, 2009.

[76] A. Meixner and D. Sorin. "Dynamic Verification of Memory Consistency in Cache-Coherent Multithreaded Computer Architectures". *IEEE Transactions on Dependable and Secure Computing*, 6:18–31, 2009.

[77] T. L. Milo D. Ercegovac. *"Digital Arithmetic"*. Morgan Kaufmann, 2003.

[78] F. Moller, J. Andersen, H. Jensen, O. Olsen, and F. Fink. "PSEUDEC: implementation of the computation-intensive PARTRAN functionality using a dedicated online CORDIC co-processor". In *International Conference on Acoustics, Speech, and Signal Processing*, pages 3207–3210, 1995.

[79] K. Molnar, C.-Y. Ho, D. Staver, B. Davis, and R. Jerdonek. "A 40 MHz 64-bit floating-point co-processor ". In *IEEE International Solid-State Circuits Conference*, 1989.

[80] R. K. Montoye, E. Hokenek, and S. L. Runyon. "Design of the IBM RISC System/6000 floating-point execution unit". *IBM Journal of Research and Development*, 34:59–70, 1990.

[81] J. Moran, I. Rios, and J. Meneses. *"Signed Digit Arithmetic on FPGAs"*. Abindon EE&CS books, 1994.

[82] J. Muller. *"Elementary Functions: Algorithms and Implementation"*. Cambridge, MA: Birkhauser, 1997.

[83] J.-M. Muller. "Discrete Basis and Computation of Elementary Functions ". *IEEE Transactions on Computers*, C-34:857–862, 1985.

[84] T. Nakayama, H. Harigai, S. Kojima, H. Kaneko, H. Igarashi, T. Toba, Y. Yamagami, and Y. Yano. "A 6.7-MFLOPS floating-point coprocessor with vector/matrix instructions". *IEEE Journal of Solid-State Circuits*, 24(5):1324–1330, October 1989.

[85] D. Narasimhan, D. Fernandes, V. Raj, J. Dorenbosch, M. Bowden, and V. Kapoor. "A 100 MHz FPGA based floating point adder". In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 3.1.1–3.1.4, 1993.

[86] C. Neri, G. Baccarelli, S. Bertazzoni, F. Pollastrone, and M. Salmeri. "Parallel hardware implementation of radar electronics equipment for a laser inspection system ". *IEEE Nuclear Science Symposium Conference Record*, 3:1352–1356, 2004.

[87] A. Nielsen, D. Matula, C. Lyu, and G. Even. "An IEEE compliant floating-point adder that conforms with the pipeline packet-forwarding paradigm". *IEEE Transactions on Computers*, 49(1):33–47, January 2000.

[88] M. Nouri, S. S. Ghaemmaghami, and A. Falahati. "Improved Window Based on Cosine Hyperbolic Function". pages 8–13, 2011.

[89] Y. Oshima, B. Sheu, and S. Jen. "High-speed memory architectures for multimedia applications ". *IEEE Circuits and Devices Magazine*, 13:8–13, 1997.

[90] A. Paidimarri, A. Cevrero, P. Brisk, and P. Ienne. "FPGA Implementation of a Single-Precision Floating-Point Multiply-Accumulator with Single-Cycle Accumulation". In *IEEE Symposium on Field Programmable Custom Computing Machines*, pages 267–270, 2009.

[91] R. Parthasarathi, E. Raman, K. Sankaranarayanan, and L. Chakrapani. "A Reconfigurable Co-Processor for Variable Long Precision Arithmetic Using Indian Algorithms". In *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 71–80, 2001.

[92] S. Paul, N. Jayakumar, and S. P. Khatri. "A Hardware Approach for Approximate, Efficient Logarithm and Antilogarithm Computations". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17:269–277, 2009.

[93] F. Philipp, F. Samman, and M. Glesner. "Design of an autonomous platform for distributed sensing-actuating systems ". In *22nd IEEE International Symposium on Rapid System Prototyping*, pages 85–90, 2011.

[94] J.-A. Pineiro, M. Ercegovac, and J. Bruguera. "Algorithm and Architecture for Logarithm Exponential, and Powering Computation". *IEEE Transactions on Computers*, 53:1085–1096, 2004.

[95] R. Pinkham, D. Redwine, and F. V. T. H. D. Anderson. "A high speed dual port memory with simultaneous serial and random mode access for video applications". *IEEE Journal of Solid-State Circuits*, 19:999–1007, 1984.

[96] P. PooGyeon and T. Kailath. "Square-root Bryson-Frazier smoothing algorithms". *IEEE Transactions on Automatic Control*, 40:761–766, 1995.

[97] A. Psomoulis, N. Cazajus, D. Dandouras, H. B. M. Gangloff, and E. Sarris. "Development of an innovative, two-processor data processing unit for the magnetospheric imaging instrument onboard the Cassini mission to Saturn. I. Hardware architecture ". *IEEE Transactions on Geoscience and Remote Sensing*, 37:1980–1996, 1999.

[98] R. Raafat, A. Abdel-Majeed, R. Samy, T. ElDeeb, Y. Farouk, M. Elkhouly, and H. Fahmy. "A decimal fully parallel and pipelined floating point multiplier". In *Asilomar Conference on Signals, Systems and Computers*, pages 1800–1804, 2008.

[99] S. Radhakrishnan, S. Chinthamani, and C. Kai. "The Blackford Northbridge Chipset for the Intel 5000". *IEEE Micro*, 27:22–23, 2007.

[100] S. Raman, V. Pentkovski, and J. Keshava. "Implementing streaming SIMD extensions on the Pentium III processor". *IEEE Micro*, 20:47–57, 2000.

[101] C. Rowen, M. Johnson, and P. Ries. "The MIPS R3010 floating-point coprocessor". *Micro, IEEE*, 8:53–63, June 1985.

[102] R. Sarmiento, V. de Armas, J. Lopez, J. Montiel-Nelson, and A. Nunez. "A CORDIC processor for FFT computation and its implementation using gallium arsenide technology ". *IEEE Transactions on very large scale integration systems*, 6:18–30, 1998.

[103] K. Sarrigeorgidis and J. Rabaey. "Ultra low power CORDIC processor for Wireless communication algorithm". *Journal of VLSI Signal Processing*, pages 115–130, 2004.

[104] T. Sasaki, Y. Ichikawa, T. Hironaka, T. Kitamura, and T. Kondo. "Evaluation of low-energy and high-performance processor using variable stages pipeline technique". *Computers & Digital Techniques*, 2:230–238, 2008.

[105] M. S. Schmookler and K. J. Nowka. "Leading Zero Anticipation and Detection – A Comparison of Methods". In *Proceedings. 15th IEEE Symposium on Computer Arithmetic*, number 7–12, 2001.

[106] M. Schulte and J. Stine. "Approximating elementary functions with symmetric bipartite tables". *IEEE Transactions on Computers*, pages 842–847, 1999.

[107] M. J. Schulte and E. E. J. Swartzlander. "A family of variable-precision interval arithmetic processors ". *IEEE Transactions on Computers*, 49:387–397, 2000.

[108] H. Shen-Fu, H. Yu-Hen, and J. Tso-Bing. "A memory-efficient and high-speed sine/cosine generator based on parallel CORDIC rotations" . *IEEE Signal Processing Letters*, 11:152–155, 2004.

[109] K. Shiann-Rong, W. Jiun-Ping, and H.-Y. Hong;. "Variable-Latency Floating-Point Multipliers for Low-Power Applications". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18:1493–1497, 2010.

[110] A. Skaf and A. Guyot. "SAGA: the first general-purpose on-line arithmetic coprocessor". In *Proceedings of the 8th International Conference on VLSI Design*, pages 146–149, 1995.

[111] B. Song and I. Lee. "A digital FM demondulator for FM, TV, and wireless". *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 42:821–825, 1995.

[112] T.-Y. Sung, Y.-S. Shieh, C.-W. Yu, and H.-C. Hsin. "A High-Efficiency Vector Interpolator Using Redundant CORDIC Arithmetic in Power-Aware 3-D Graphics Rendering". In *Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2006.

[113] H. Suzuki, H. Morinake, H. Makino, Y. Nakase, K. Mashiko, and T. Sumi. "Leading-Zero Anticipatory Logic for High Speed Floating-Point Addition". *IEEE Journal of Solid-State Circuits*, 31(8):1157–1164, 1996.

[114] N. Takagi, T. Asada, and S. Yajima. "Redundant CORDIC methods with a constant scale factor for sine and cosine computation". *IEEE Transactions on Computers*, 40:989–995, September 1991.

[115] P. Tang. "Table-lookup algorithms for elementary functions and their error analysis". In *Proceedings., 10th IEEE Symposium on Computer Arithmetic, 1991.*, 1991.

[116] K. Teh, A. Kot, and K. Li. "Performance analysis of an FFH/BFSK product-combining receiver under multitone jamming ". *IEEE Transactions on Vehicular Technology*, 48:1946–1953, 1999.

[117] C. Tsen, S. Gonzalez-Navarro, and M. Schulte. "Hardware design of a Binary Integer Decimal-based floating-point adder". In *International Conference on Computer Design*, pages 288–295, 2007.

[118] J. Tso-Bing, H. Shen-Fu, and T. Ming-Yu. "Para-CORDIC: parallel CORDIC rotation algorithm". *IEEE Transactions on Circuits and Systems I, Regular Papers*, 51:1515–1524, 2004.

[119] L. Vachhani, K. Sridharan, and P. K. Meher. "Efficient FPGA Realization of CORDIC with Application to Robotic Exploration". *IEEE Trans. Industrial Electronics*, 56(12):4915–4929, Dec. 2009.

[120] J. Valls. "Evaluation of CORDIC Algorithms for FPGA Design". *Journal of VLSI Signal Processing*, 32:207–222, 2002.

[121] S. Vangal, Y. Hoskote, D. Somasekhar, V. Erraguntla, J. Howard, G. Ruhl, V. Veeramachaneni, D. Finan, S. Mathew, and N. Borkar. "A 5 GHz floating point multiply-accumulator in 90 nm dual VT CMOS". In *IEEE International Solid-State Circuits Conference*, pages 334–497, 2003.

[122] S. R. Vangal, Y. V. Hoskote, N. Y. Borkar, and A. Alvandpour. "A 6.2-Gflops Floating-Point Multiply-Accumulator With Conditional Normalization". *IEEE journal of solid-state circuits*, 41:2314–2323, October 2006.

[123] J. Villalba, E. L. Zapata, E. Antelo, and J. D. Bruguera. "Radix-4 Vectoring CORDIC Algorithm and Architectures". *The Journal of VLSI Signal Processing*, 19(2):127–147, 1998.

[124] J. Volder. "The CORDIC trigonometric computing technique". *IRE Trans. Electron. Compute.*, EC-8:330–334, 1959.

[125] J. Volder. "The birtth of CORDIC". *The Journal of VLSI Signal Processing*, 25(2):101–105, 2000.

[126] J. Walther. "A unified algorithm for elementary functions". *Spring Joint Computer Conf.*, pages 379–385, 1971.

[127] K. Wang, C. Bryant, M. Carlson, T. Elmer, A. Harris, M. Garcia, C. S. Hui, C. K. Leong, B. Reynolds, R. Tang, L. Weber, J. Wenzel, G. Wilson, and M. Becker. "Designing the MPC105 PCI bridge/memory controller ". *IEEE Micro*, 15:44–49, 1995.

[128] R. Wei, M. Jin, J. Xia, Z. Xie, and H. Liu. "Reconfigurable Parallel VLSI Co-Processor for Space Robot Using FPGA ". In *IEEE International Conference on Robotics and Biomimetics*, 2006.

[129] G. Wolrich, E. McLellan, L. Harada, J. Montanaro, and R. Yodlowski. "A high performance floating point coprocessor". *IEEE Journal of Solid-State Circuits*, 19(5):690–696, October 1984.

[130] T.-M. Wu. "A Suboptimal Maximum-Likelihood Receiver for FFH/BFSK Systems With Multitone Jamming Over Frequency-Selective Rayleigh-Fading Channels". *IEEE Transactions on Vehicular Technology*, 57:1316–1322, 2008.

[131] Xilinx. "Floating-point operator v3.0". In *Xilinx 3rd*, 2006.

[132] H. Yamada, T. Hotta, T. Nishiyama, F. Murabayashi, T. Yamauchi, and H. Sawamoto. "A 13.3ns double-precision floating-point ALU and multiplier". In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 466–470, 1995.

[133] M. Ye, T. Liu, Y. Ye, G. Xu, and T. Xu. "FPGA Implementation of CORDIC-Based Square Root Operation for Parameter Extraction of Digital Pre-Distortion for Power Amplifiers". In *The 6th International Conference on Wireless Communications Networking and Mobile Computing*, 2010.

[134] R. Yu and G. Zyner. "167 MHz radix-4 floating point multiplier". In *12th Symposium on Computer Arithmetic*, pages 149–154, 1995.

[135] W. Yun and W. Jinkuan. "Fast Square-Root Detection Algorithm for V-BLAST ". In *International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1340–1343, 2007.

[136] J. Zhou, Y. Dou, Y. Lei, J. Xu, and Y. Dong. "Double Precision Hybrid-Mode Floating-Point FPGA CORDIC Co-processor". In *IEEE International Conference on High Performance Computing and Communications*, 2008.

List of Own Publications

Related Publications

[137] F. A. Sammany, P. Surapong, C. Spies, and M. Glesner. "Floating-point-based Hardware Accelerator of a Beam Phase-Magnitude Detector and Filter for a Beam Phase Control System in a Heavy-Ion Synchrotron Application". In *International Conference on Accelerator and Large Experimental Physics Control Systems*, pages 683–686, 2011.

[138] P. Surapong and M. Glesner. "On-chip efficient Round-Robin scheduler for high-speed interconnection". In *22nd IEEE International Symposium on Rapid System Prototyping*, pages 199–202, 2011.

[139] P. Surapong and M. Glesner. "Pipelined Floating-Point Architecture for a Phase and Magnitude Detector based on CORDIC". In *International Conference on Field Programmable Logic and Application*, pages 382–384, 2011.

[140] P. Surapong, M. Glesner, and H. Klingbeil. "Implementation of realtime pipeline-folding 64-tap filter on FPGA". In *PhD-Forum: PhD Research in Microelectronics and Electronics (PRIME)*, pages 1–4, 2010.

[141] P. Surapong, M. Glesner, and H. Klingbeil. "TDM switch-based crossbar architecture for SoC on FPGA ". In *PhD-Forum: PhD Research in Microelectronics and Electronics (PRIME)*, pages 1–4, 2010.

[142] P. Surapong, F. Philipp, F. A. Samman, and M. Glesner. "Improvement of Standard and Non-Standard Floating-Point Operators". *ECTI Transaction Computer and Information Technology*, 6(1):9–22, May 2012.

[143] P. Surapong, F. A. Samman, and M. Glesner. "Reconfigurable streaming processor core with interconnected floating-point arithmetic units for multicore adaptive signal processing systems ". In *6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip*, pages 1–6, 2011.

[144] P. Surapong, F. A. Samman, and M. Glesner. "Design and Analysis of Extension-Rotation CORDIC Algorithms based on Non-Redundant Method". *International*

Journal of Signal Processing, Image Processing and Pattern Recognition, 5(1):65–84, March 2012.

[145] P. Surapong, F. A. Samman, M. Glesner, and S. Surachat. "Design and Evaluation of a Floating-Point Division Operator based on CORDIC Algorithm". In *International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 1–4, 2012.

[146] P. Surapong, S. Surachat, and M. Glesner. "An Interleaving Switch-based Crossbar Architecture for MP SoC on FPGA" . In *International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 188–192, 2010.

Unrelated Publications

[147] S. Choomchuay, P. Surapong, and K. Hadkhuntod. "An On-chip Data Storage With ElGamal Elliptic Curves Cryptosystem". In *Int. Symposium on Communication and Information Technology*, pages 423–426, 2002.

[148] S. Choomchuay, P. Surapong, and S. Phathumvanh. "A Compact 32-bit Architecture for an AES System". *ECTI Transaction Computer and Information Technology*, 1(1):24–29, 2005.

[149] P. Surapong and S. Choomchuay. "Composite Field Bit Serial-Parallel Multiplier". In *Proc. of Information and Computer Engineering Workshop*, pages 65–69, 2002.

[150] P. Surapong and S. Choomchuay. "An Architecture for a SHA-1 Applied for DSA". In *Electrical Eng./Electronics, Communications, Computer and Information Technology Conference*, pages 133–136, 2004.

[151] P. Surapong, S. Choomchuay, and K. Hadkhuntod. "An Application of ELGamal Elliptic Curves Cryptosystems in Smart Card". In *Electrical Engineering Conference, EECON-25*, pages 46–50, 2002.

[152] P. Surapong, P. Noo-intara, and S. Choomchuay. "SHA-1 With On The Fly Round-Word Computation". In *Electrical Eng./Electronics, Communications, Computer and Information Technology Conference*, 2004.

[153] P. Surapong, S. Phathumvanh, and S. Choomchuay. "A 32 Bits Architecture for an AES System". In *IEEE Int. Symposium on Communications and Information Technologies*, pages 70–73, 2004.

Index

Curriculum Vitae

Personal Data :

Surname	: Pongyupinpanich
Name	: Surapong
Birth date	: 17 June 1976
Location	: Prachinburi, Thailand

School Education :

Year	*School level*	*School name*
$1981 - 1987$	Primary school	Mary-Withaya School, Prachinburi, Thailand
$1989 - 1991$	Secondary school	Mary-Withaya School, Prachinburi, Thailand
$1992 - 1995$	High school	Rajamakala Institute of Technology, Nonthaburi, Thailand

Higher Education :

Year	*Degree*	*Institute*
$1996 - 1998$	Bachelor Degree	King Mongkuts Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand.
$1999 - 2002$	Master Degree	King Mongkuts Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand.
$2008 - 2012$	Doctoral Degree	Technische Universität Datmstadt (TUD), Germany

Work Experience :

Year	*Position*	*Organization*s
$2000 - 2001$	Engineer	LPG Engineering Company Limited, Bangkok, Thailand
$2001 - 2003$	Researcher	King Mongkuts Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand.
$2003 - 2004$	Researcher	Naresuan University, Phayao, Thailand
$2004 - 2005$	Senior Engineer	Thai Smart Card Co.,Ltd., Bangkok, Thailand
$2005-$Current	Researcher	Ramkhamhaeng University, Bangkok, Thailand